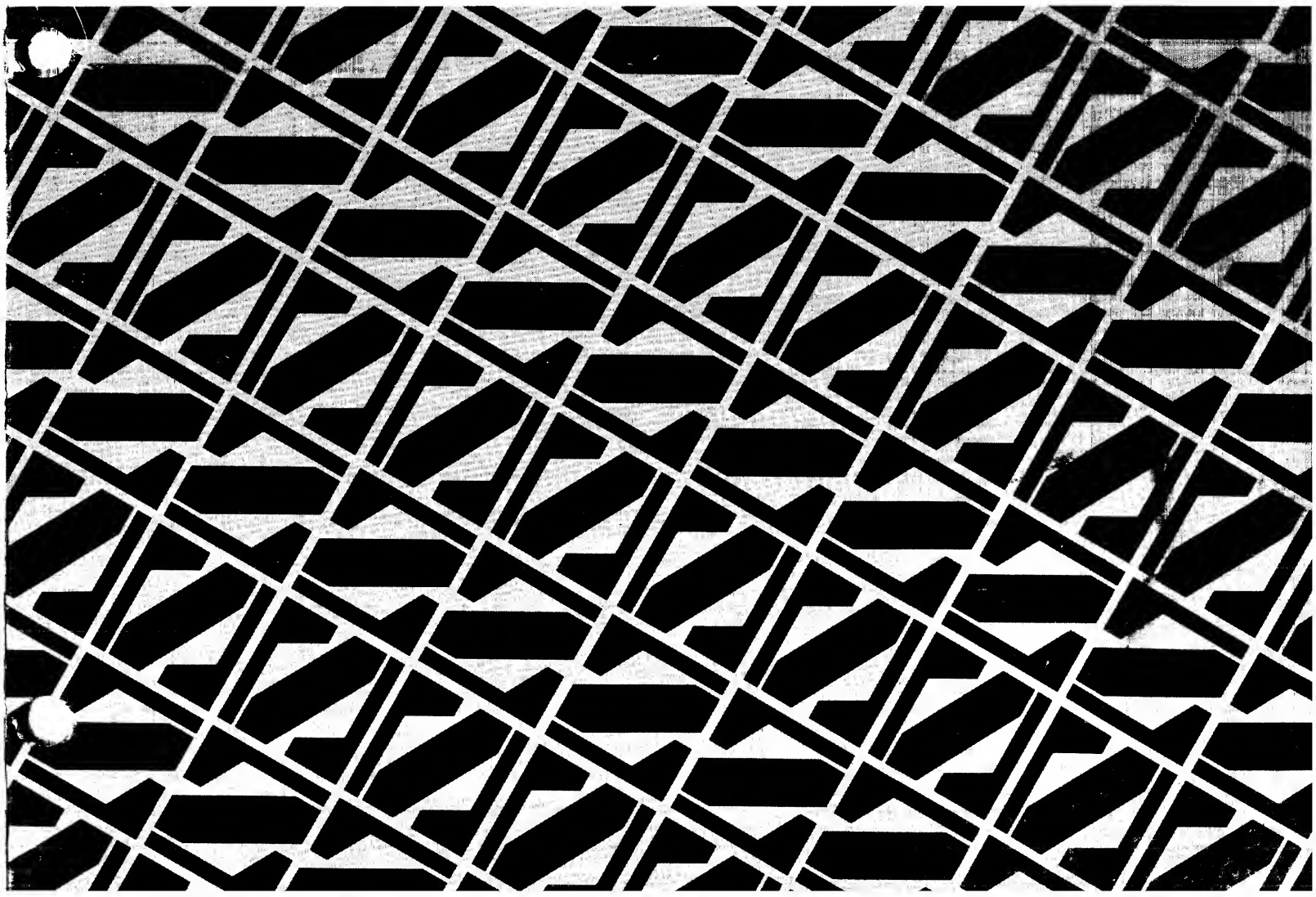
National Semiconductor
Order No. IMP-16C/921C
Pub. No. 4200021C

IMP-16C Application Manual



Order Number IMP-16C/921C
Publication Number 4200021C

Integrated MicroProcessor-16C

IMP-16C
APPLICATION MANUAL

January 1974

2900 Semiconductor Drive
Santa Clara, California 95051

PREFACE

The IMP-16C Application Manual provides information required for a user to become familiar with the IMP-16C microprocessor functional and logic circuits, instruction set, general input/output interfacing, and a general system verification. With this information, the user may adapt the IMP-16C microprocessor to his particular application(s).

This issue, order number IMP-16C/921C, pertains to the IMP-16C/200 and IMP-16/300 microprocessors. The IMP-16C/200 is a pin-compatible version of the earlier IMP-16C card, with the layout changed to accommodate a second Control Read Only Memory (CROM). This second CROM may be supplied with an extended instruction set, or the second CROM can be customized for special applications. The IMP-16C/200 is supplied with only one CROM (programmed for the basic instruction set) and with an empty CROM socket. The IMP-16C/300 has two CROMs, the second CROM programmed for the extended instruction set and inserted in the second CROM socket. This is the only difference between the two cards.

The IMP-16C Interfacing Guide (formerly issued as publication number 4200035A) is presented as Supplement 1 to this manual.

The material in this manual is for information purposes only and is subject to change without notice.

Copies of this publication and other National Semiconductor publications may be obtained from the sales offices listed on the back cover.

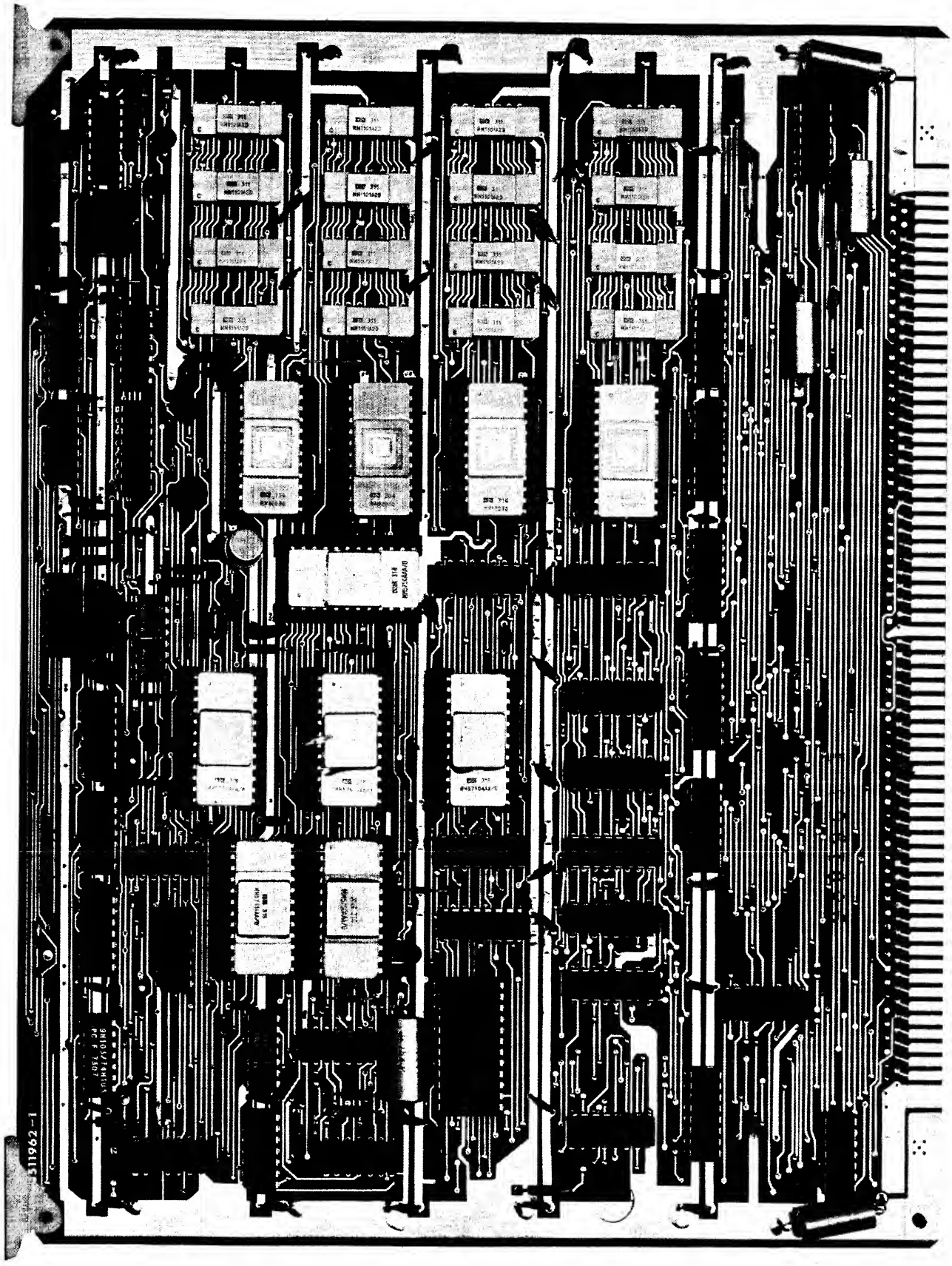
CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

ILLUSTRATIONS

TABLES



NS00145

IMP-16C Card

Chapter 1
GENERAL INFORMATION

1.1 IMP-16C CONFIGURATION

The IMP-16C is a 16-bit parallel processor. It is packaged on an 8½-by-11-inch printed wiring card, which is shown on the facing page. A 144-pin connector is located on the edge of the card for connecting the IMP-16C circuits to interfacing units.

The major functional units of the IMP-16C are shown in figure 1–1 and are composed of the following:

- Central Processing Unit (CPU)
- Clock Generators
- Input Multiplexer
- Data Buffer
- Control Flags
- Conditional Jump Multiplexer
- On-Card Memory
- Address Latches

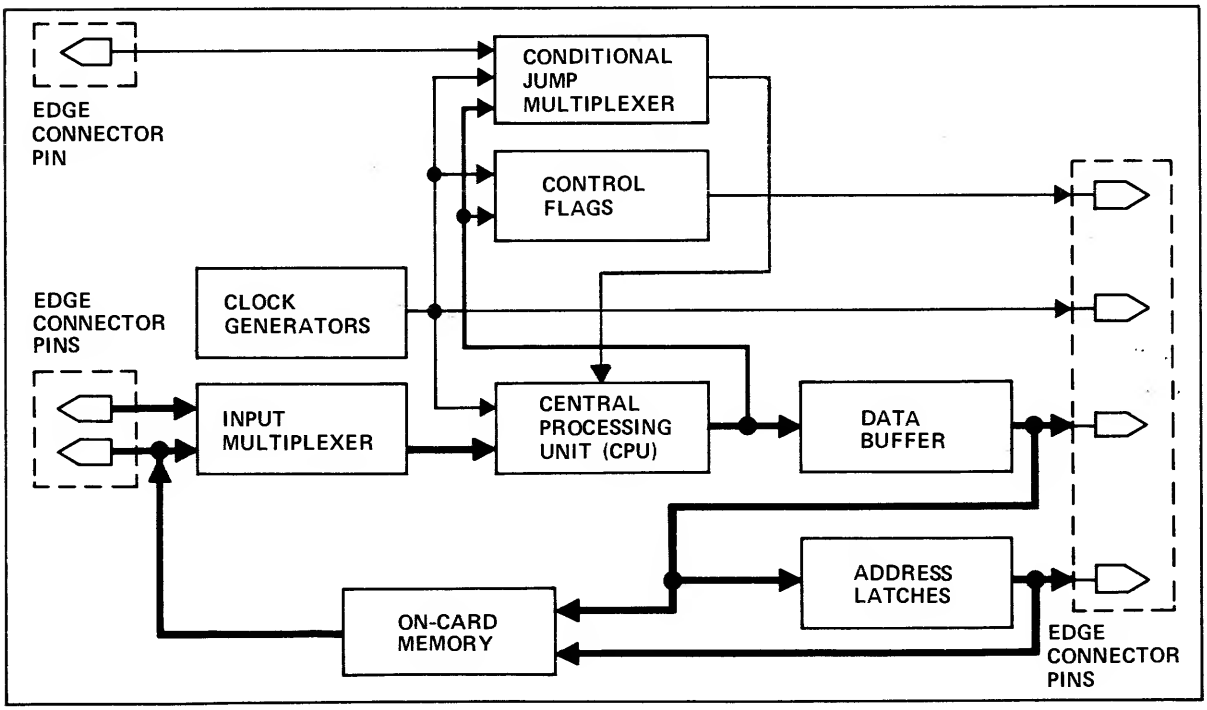


Figure 1–1. IMP-16C Major Functional Units

The CPU is configured around the National Semiconductor GPC/P (General-Purpose Controller/Processor) MOS/LSI devices, as shown in the simplified block diagram of figure 1–2. The MOS/LSI devices consist of one CROM (Control Read Only Memory) and four RALUs (Register and Arithmetic Logic Units). Each RALU handles 4 bits, and a 16-bit unit is formed by connecting four RALUs in parallel. A 4-bit-wide control bus is used by the CROM to communicate most of the control information to the RALUs. The CPU includes provision for adding a second CROM for an optional extended instruction set.

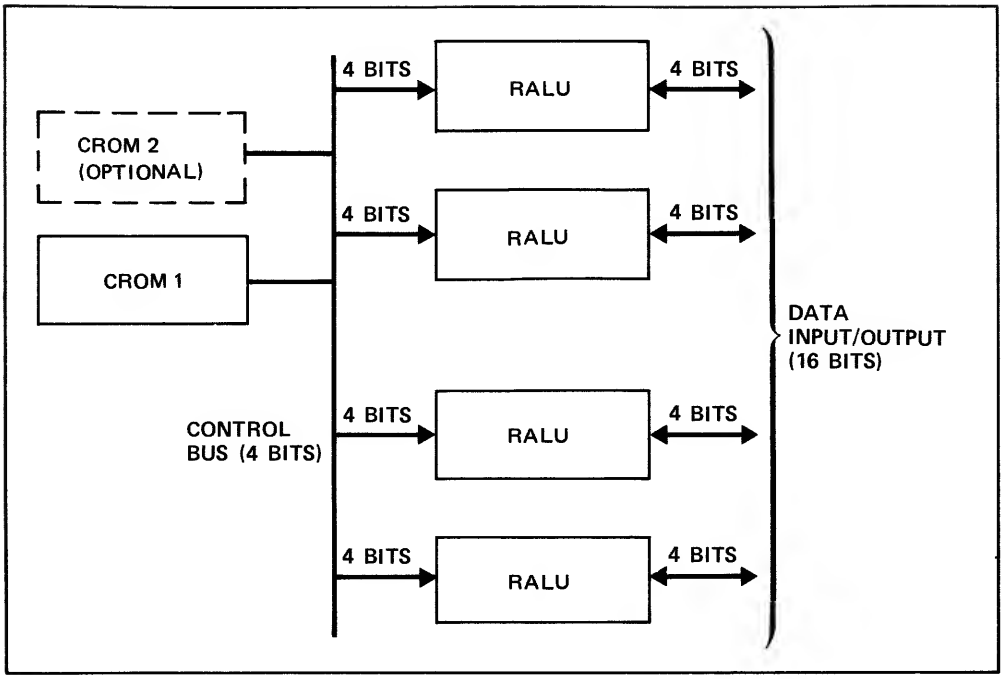


Figure 1–2. IMP-16C CPU Components

The Clock Generator provides the MOS clock drivers and CPU timing signals. The system clock is distributed outside of the IMP-16C for synchronization of peripheral units with the IMP-16C.

External to the MOS/LSI circuits but still within the IMP-16C are control flags for both the IMP-16C and external interfacing circuits. These control flags are in addition to the status flags that are internal to the RALUs. The status flags are discussed in the description of the CPU in chapter 2. The control flags are discussed in chapter 3 under the control flag instructions. Conditional branches are selected by the Conditional Jump Multiplexer. These branch conditions are discussed in chapter 3 under the Branch-On-Condition (BOC) instruction.

Data from the user's peripheral devices and add-on memory are received by the Input Multiplexer. Data from the On-Card Memory are also processed through the Input Multiplexer en route to the Central Processing Unit.

Output data are made available from the 16-bit Data Buffer via the card-edge connector to the user's peripheral devices and add-on memory. A 16-bit address bus is also brought out to the card-edge connector for addressing both add-on memory and peripheral devices.

The memory on the IMP-16C card consists of 256 words of read/write memory and sockets for 512 words of PROM or ROM. A maximum of 65,536 words may be addressed.

Chapter 2 contains a more-detailed functional description of the IMP-16C. The instruction set is explained in chapter 3. Chapter 4 presents a circuit-level description.

1.2 IMP-16C OPERATIONAL FEATURES

Word Length	16 bits
Instruction Set	43 in basic instruction set; 17 optional instructions available with extended set (macroinstructions implemented by CPU-resident microprogram)
Arithmetic	Parallel, binary, fixed point, twos complement

Memory	256 16-bit words of semiconductor read/write memory
	Sockets for 512 16-bit words of semiconductor read-only memory
	Capable of addressing 65,536 16-bit words
Addressing	Page size of 256 words. For direct and indirect modes:
	• Absolute • Relative to Program Counter • Relative to Accumulator 2 (indexed) • Relative to Accumulator 3 (indexed)
Typical Instruction-Execution Speeds	Register-to-register addition – 4.55 microseconds
	Memory-to-register addition – 7.7 microseconds
	Register input/output – 10.15 microseconds
Input/Output and Control	16-bit data-input port
	16-bit data-output bus
	16-bit address bus
	6 general-purpose flags
	4 general-purpose jump-condition inputs
	1 general interrupt input
	1 control panel interrupt input

1.3 POWER AND ENVIRONMENTAL REQUIREMENTS

The IMP-16C card contains both standard TTL integrated circuits and MOS/LSI components. These circuits require +5-volt and –12-volt 5% regulated dc supplies. In addition, the read/write memory on the card requires a –9-volt supply, which can be developed from the –12-volt source. Typical current requirements are given below for IMP-16C operation at maximum speed; however, when operating at lower speeds, the requirements on the –12-volt and –9-volt supplies are a few percent less than indicated below.

Voltage and Current	+5 volts at 2.25 amperes
	–12 volts at 0.5 amperes
	–9 volts at 0.6 amperes

When the IMP-16C on-card memory is disabled (see appendix C), the –9-volt power supply is not needed.

NOTE

Refer to 8.7 of this manual for operating procedures; these provide the prerequisite instructions for proper use of the IMP-16C.

Temperature	Operating – 0 to 70° C
	Storage – -20 to 100° C
Humidity	Maximum of 90% relative humidity without condensation

Chapter 2
IMP-16C FUNCTIONAL DESCRIPTION

2.1 FUNCTIONAL UNITS AND DATA FLOW

A simplified block diagram showing the data flow among the major functional units of the IMP-16C is given in figure 2–1. The main components of the Central Processing Unit (CPU) and the Input/Output Section are shown. The Clock Generator and Timing Control, the Control Flags and Conditional Jump Multiplexer, and the Memory are not detailed at this time; they are discussed in detail in chapter 4.

2.1.1 INPUT DATA

Incoming data from peripheral units and data from memory are received by the Input Multiplexer and made available directly to the four Register and Arithmetic Logic Units (RALUs) and to the Data Buffer over a 16-bit data bus. A 4-bit bidirectional bus connects each RALU to the data bus.

2.1.2 DATA AND ADDRESS TRANSFERS

Both addresses and data are transferred from the Data Buffer over the 16-bit Buffered Data Out (BDO) bus. This bus is brought out to pins on the card-edge connector for transferring data to peripheral units and off-card memory. The on-card memory receives data directly from the BDO bus. Addresses are loaded into the Address Latches, from which they are brought out to another set of pins on the card-edge connector. Addresses are also routed to the on-card memory directly from the Address Latches.

2.1.3 CENTRAL PROCESSING UNIT (CPU) COMMUNICATIONS

Communications between the CROM(s) and the RALUs are effected over a 4-bit control bus. Because these units constitute the main part of the control and data processing capability of the IMP-16C, a comprehensive description of them follows.

Controlling the operations of the CPU is the Control Read-Only Memory (CROM). The control is effected by routines that constitute the microprogram stored in the Read-Only Memory of the CROM. The microprogram effects the implementation of macroinstructions that comprise the IMP-16C instruction set, with an expanded set available in a second CROM.

Because the actual data processing takes place in the four 4-bit Register and Arithmetic Logic Units (RALUs), they are described first (2.2) to establish a basis for their control by the Control Read-Only Memory (CROM) and also for the descriptions of the IMP-16C instruction set described in chapter 3.

2.2 REGISTER AND ARITHMETIC LOGIC UNITS (RALUs)

Four 4-bit RALUs constitute the Arithmetic Section shown in figure 2–2. This section includes the following major units:

- Input/Output Multiplexer
- Last-In/First-Out Stack (LIFOS)

- 16 Status Flags, storable and retrievable as a 16-bit register
  - Link Flag (L)
  - Carry Flag (CY)
  - Overflow Flag (OV)
  - 13 General-Purpose Flags (two of which are directly accessible to user at the edge connector)
- Program Counter (PC)
- Memory Data Register (MDR)
- Memory Address Register (MAR)
- 4 Accumulators (AC0, AC1, AC2, and AC3)
- Arithmetic and Logic Unit (ALU)
- Shifter
- Buses

The functions of these units are described in the following paragraphs.

2.2.1 INTERNAL BUSES

Three buses are internal to the RALU: A-bus, B-bus, and R-bus. These buses are described below.

A-Bus (Operand Bus). The contents of all RALU registers may be loaded onto the A-bus; data from the top of the Last-In/First-Out (LIFO) stack and from the RALU Status Flags (combined as a 16-bit word) may be loaded as operands. During such loading on the A-bus, the data may be complemented under control of the CROM. The contents of the A-bus may be gated through the ALU and Shifter to the R-bus or out of the RALU to the IMP-16C data bus through the Input/Output Multiplexer.

B-Bus (Operand Bus). The contents of all RALU registers may be loaded onto the B-bus. The contents of the B-bus may be loaded only into the Arithmetic and Logic Unit (ALU).

R-Bus (Result Bus). The R-bus serves to transfer the results of ALU operations to any RALU register, the LIFOS, and the RALU flags.

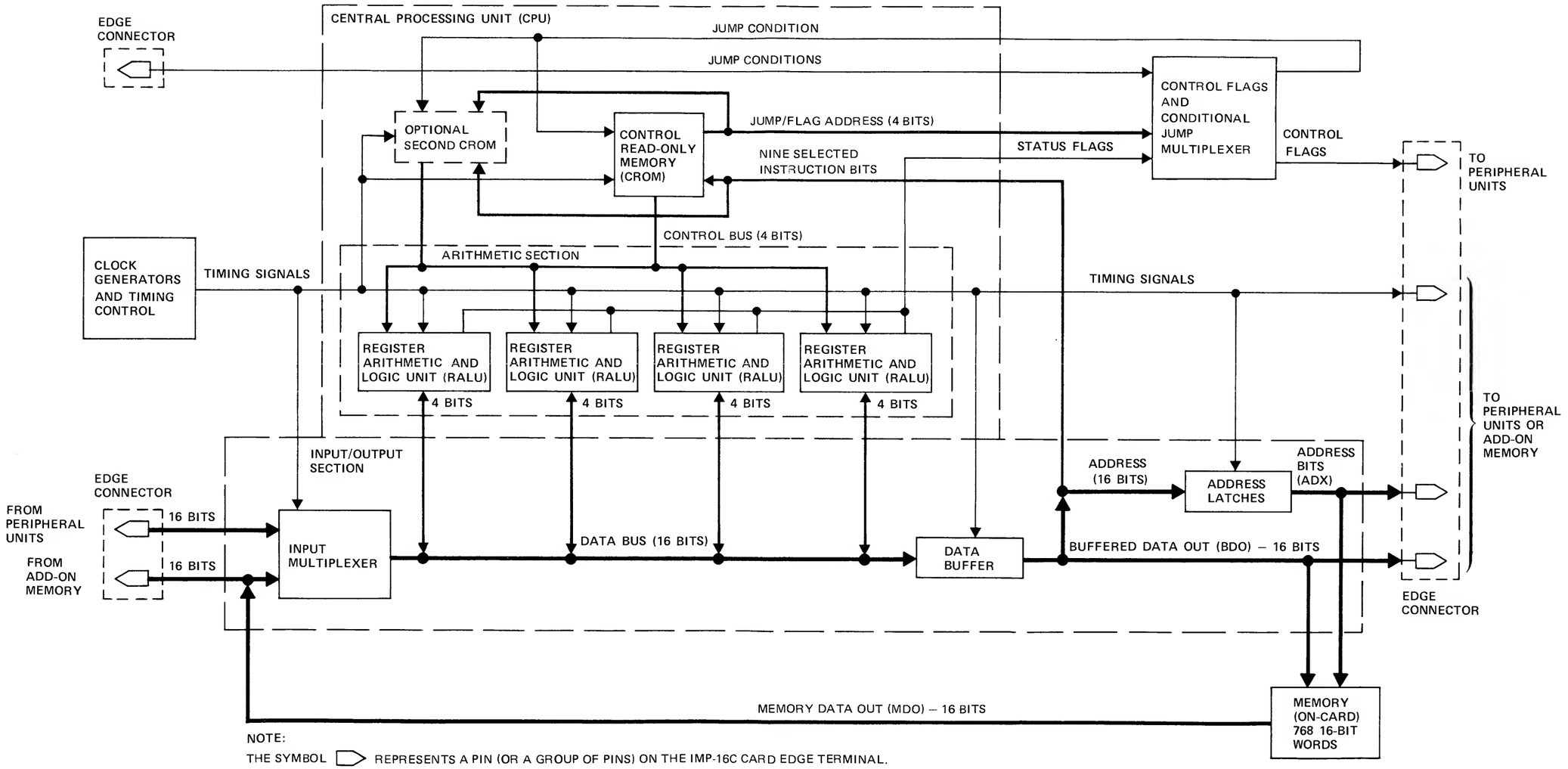
2.2.2 LAST-IN/FIRST-OUT STACK (LIFOS)

Each RALU has a stack that operates on a last-in/first-out basis. The stack is 16 words high and is accessible through the top location. The 16-bit-per-word stack of figure 2–2 is contained in the four 4-bit RALUs comprising the Arithmetic Section. A 16-bit data word is entered via the R-bus and retrieved via the A-bus. As a word is entered into the top location of the stack, the 16 bits in the top location are pushed down one level, and the entered bits occupy the top location. The contents of each lower level are replaced by the contents of the next higher location, and the contents of the bottom location are lost. The reverse process occurs when a word is retrieved from the stack; in this case, zeros are entered into the bottom location.

The stack is used primarily for saving of status during interrupts and for temporary storage of subroutine return addresses. It may also be used to temporarily store data using the appropriate instructions, described in chapter 3.

2.2.3 RALU FLAGS

There are 16 RALU status flags. These flags may be pushed onto the stack for temporary storage during interrupt processing. The flags may be transferred back into their respective flag flip-flops after completion of the interrupt service. Whenever the status flags are manipulated, the entire complement of flags is configured as a 16-bit register; the L (link), CY (carry), and OV (overflow) flags are the first, second, and third most significant bits, respectively, and the remaining general-purpose flags comprise the 13 less significant bits. The CY or the OV flags may be selected for output on the CYOV line from the RALUs, under control of the SEL control flag. For SEL control flag usage, see 4.3.



NS00123

Figure 2–1. IMP-16C Simplified Block Diagram

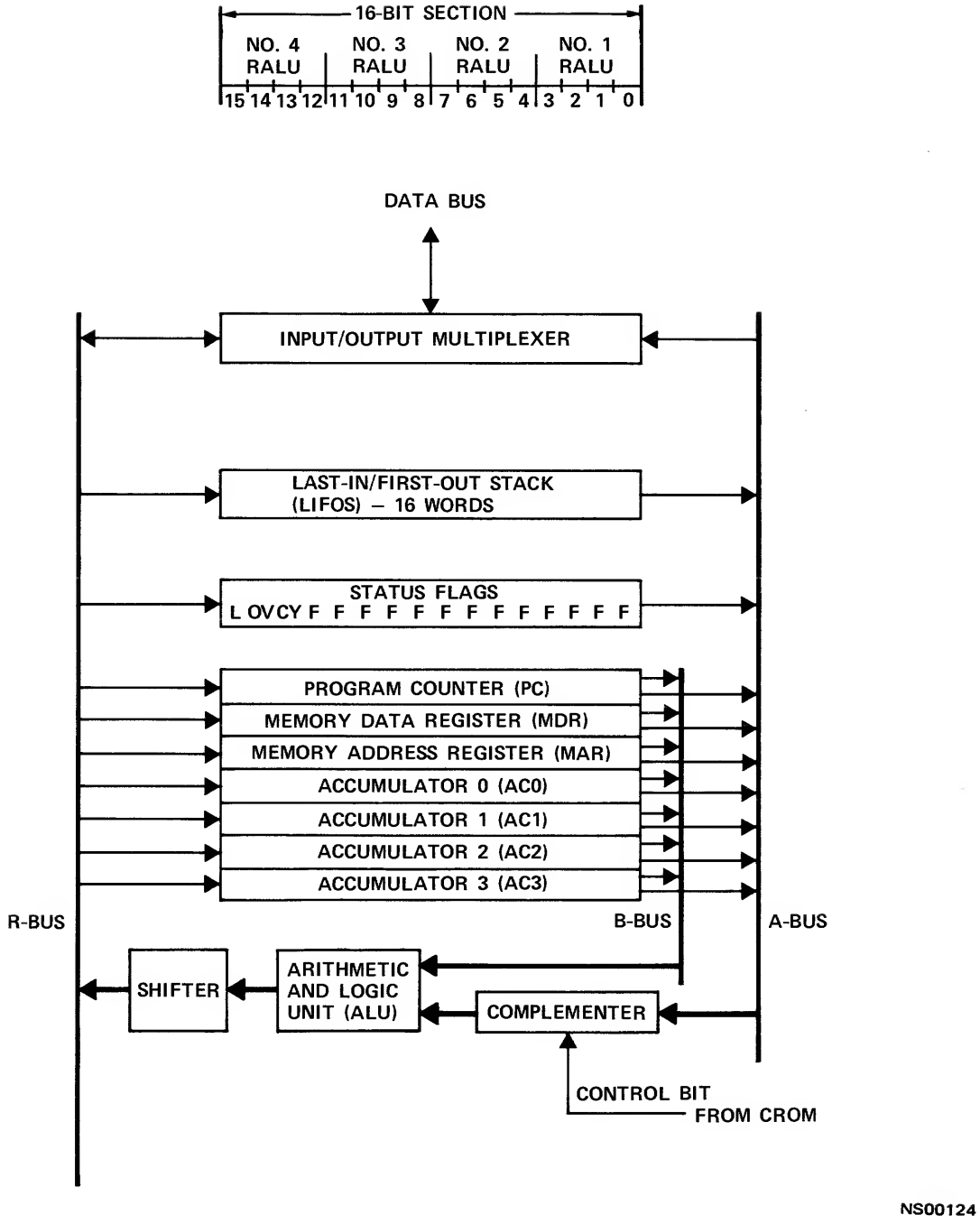


Figure 2–2. IMP-16C 16-Bit Arithmetic Section

2.2.4 PROGRAM COUNTER (PC)

The Program Counter (PC) holds the address of the next instruction to be executed. It is incremented by 1 immediately following the fetching of each instruction during execution of the current instruction. When there is a branch to another address in the main memory, the branch address is set into the Program Counter. A skip instruction merely increments the Program Counter by 1, thus causing the one instruction to be skipped.

2.2.5 MEMORY DATA REGISTER (MDR) AND MEMORY ADDRESS REGISTER (MAR)

The Memory Data Register (MDR) holds data transferred from the main memory to the processor, or vice versa. When fetching data, the effective address is placed in the Memory Address Register (MAR), and the fetch instruction causes the data word to be transferred from the designated main-memory location to the Memory Data Register. Conversely, when storing data in the main memory, the data word is placed in the Memory Data Register, the effective address is placed in the Memory Address Register, and the store instruction causes the data word to be transferred to the designated memory location.

2.2.6 ACCUMULATORS AC0, AC1, AC2, AND AC3

The accumulators hold operands for data manipulation during arithmetic and logical operations. Also, the result of an operation is usually stored temporarily in one of the four accumulators. Data words may be fetched from memory to the accumulator or stored from the accumulator into memory. The particular accumulator to take part in an operation is specified by the programmer in the appropriate instruction.

2.2.7 ARITHMETIC AND LOGIC UNIT (ALU), SHIFTER, AND COMPLEMENTER

The Arithmetic and Logic Unit (ALU) performs both arithmetic and logical operations: binary addition, AND, OR, and exclusive OR. Arithmetic and logical operations are effected by the microprogram that is part of the CROM. The IMP-16C performs arithmetic using the twos-complement technique. The contents of the A-bus may be selectively complemented under control of control bits from the CROM.

The output of the Arithmetic and Logic Unit is transferred to the R-bus through the Shifter and may then be stored in the stack or any of the RALU registers. The Shifter is capable of performing a single-position shift, either to the left or to the right, during each basic machine cycle.

2.2.8 INPUT/OUTPUT MULTIPLEXER

The Input/Output Multiplexer handles data, address, and instruction transfers into and out of the RALU from or into the main memory and peripheral devices on a time-multiplexed basis. As shown in figure 2–2, output data (to data bus) must be received from the A-bus, and input data passes from the Input/Output Multiplexer to the R-bus.

2.3 CONTROL AND READ-ONLY MEMORY (CROM)

Figure 2–3 is a simplified block diagram emphasizing the role of the CROM and four RALUs as part of the IMP-16C. Data buffers are provided between the CPU MOS/LSI devices and other parts of the equipment that have been implemented with TTL logic. These buffers are TRI-STATE® logic elements that permit bus-connected inputs.

The primary control of the RALU devices is accomplished over the 4-bit control bus. The control bus is time-multiplexed to transfer four 4-bit words of control information per machine cycle. The functions effected by the control bits are indicated in figure 2–4.

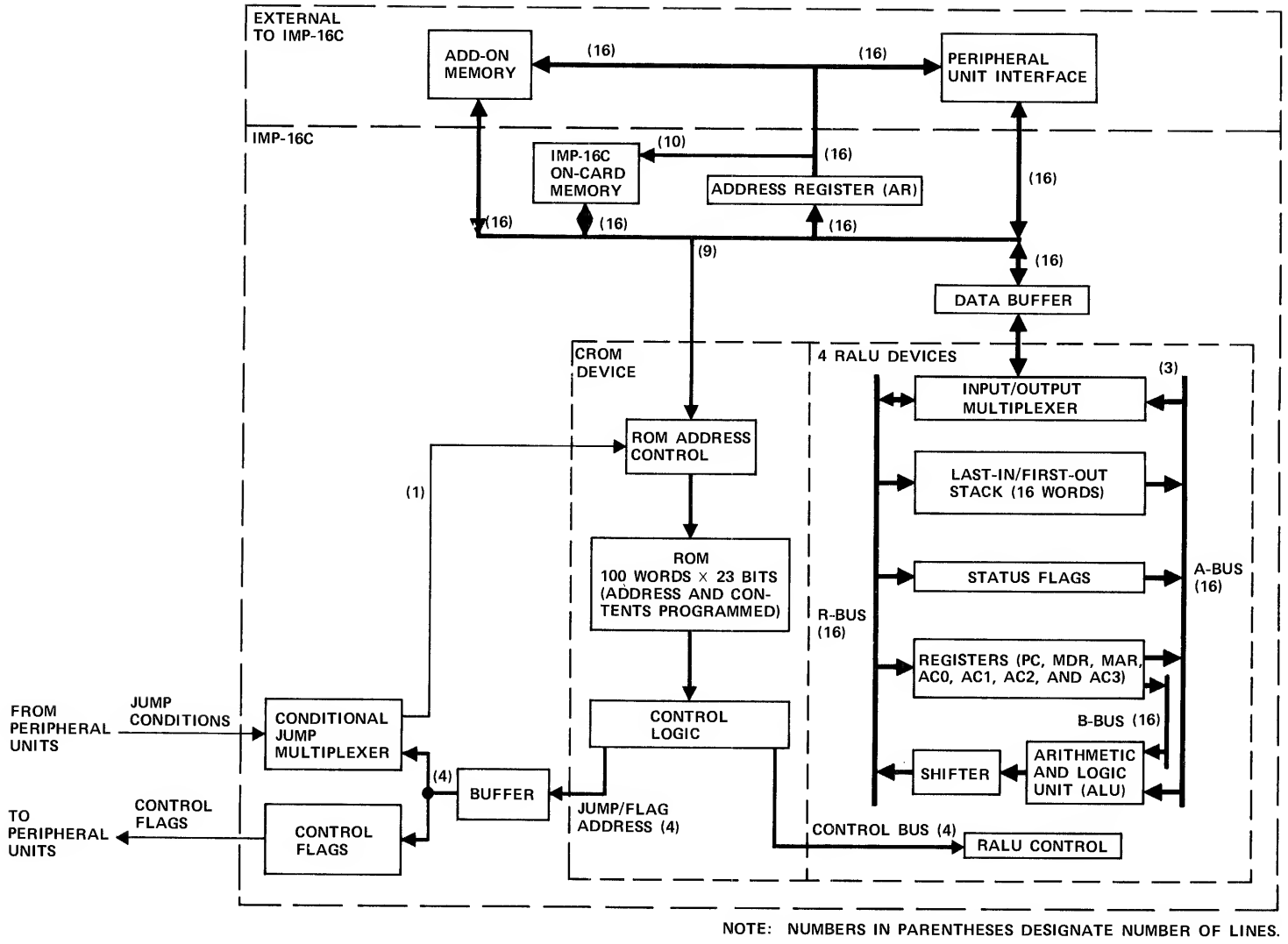
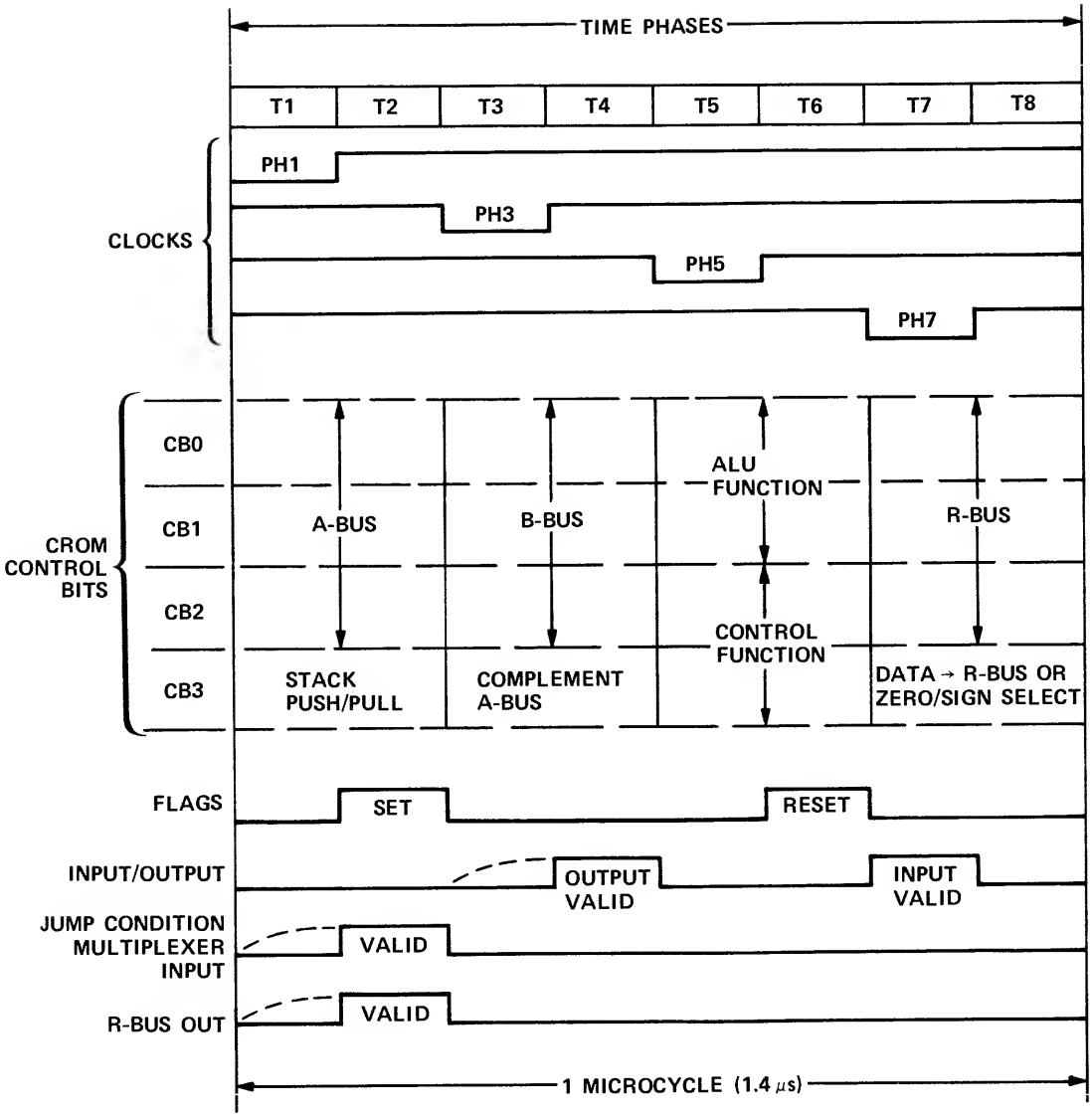


Figure 2–3. IMP-16C Control and Read-Only Memory (CROM) and Register and Arithmetic Logic Units (RALUs) Interrelations



NS00126

Figure 2–4. IMP-16C Timing Control

The CROM contains a microprogram that implements the standard instruction set. This microprogram resides in a 100-by-23-bit ROM. During an instruction fetch, the 9 most significant bits of the instruction word are transferred to the CROM; these 9 bits comprise the opcode and other pertinent control fields of an instruction word. The instruction bits are decoded, and then the ROM Address Control in the CROM directs the control sequence to an entry point in the microprogram. The sequence continues until execution of the fetched instruction is completed. Then, the CROM goes through another fetch cycle to fetch the next instruction from memory. This process is continuously repeated.

2.4 IMP-16C TIMING

The basic machine cycle of the IMP-16C consists of the execution of a single microprogram step. This cyclic time period comprises eight time intervals: T1, T2,...,T8. As indicated in the timing diagram of figure 2–4, clock pulses (phases 1, 3, 5, and 7) occur in the four clock lines at the respective odd-time periods T1, T3, T5, and T7.

2.4.1 EFFECT OF CONTROL BITS

NOTE

The effects of the CROM control bits described in 2.4.1 are presented for completeness to show their relationship to the other signals in figure 2–4. Comprehension of this description is not required to understand the operation of the IMP-16C and, for this reason, may be deferred for later reading.

The encoding of the control bits shown in figure 2–4 for the 4-line time-multiplexed control bus (between the CROM and the RALU) is described below for each of the four time phases. It is during these time phases that control information is transferred from the CROM to the RALU. Control bus lines are designated CB0 through CB3.

Phase 1 Control Bits

- If the 3-bit A-bus address field is nonzero, the contents of the RALU register designated in the macroinstruction are gated onto the A-bus.
- If the A-bus address field is zero and the push-pull control bit is "1," the LIFO Stack is pulled, and the contents of the top of the stack are gated onto the A-bus.
- If the A-bus address field is zero and the push-pull control bit is "0," then a value of zero is gated onto the A-bus.

NOTE

Pushing data onto the stack is also contingent on the push-pull control bit and is described under Phase 7 Control Bits, where the operation occurs.

Phase 3 Control Bits

- If the 3-bit B-bus address field is nonzero, then the contents of the register designated in the macroinstruction are gated onto the B-bus.
- If the B-bus address field is zero, a value of zero is gated onto the B-bus.
- The Complement A-bus bit causes the A input to the ALU to be complemented.

Phase 5 Control Bits

- The ALU bits designate a function according to table 2–1.

Table 2–1. ALU Function Bits

Code	Function
00	AND
01	XOR
10	OR
11	ADD

- The Control Function bits designate a function according to table 2–2. The no-op function applies only to the Control Function field. The expression R ← 0/SIGN means that either zeros or the sign of the less significant byte of the word being transferred to the R-bus is propagated throughout the more significant byte. (If the fourth control bit CB3 is "1" during phase 7, the sign is propagated; otherwise, zero is propagated.)

Table 2–2. Control Function Bits

Code	Function
00	No-Op (no operation for control bits only)
01	R ← 0/SIGN (zero or sign propagation)
10	LSH (Left Shift)
11	RSH (Right Shift)

Phase 7 Control Bits

- If the 3-bit R-bus field is nonzero, the contents of the R-bus are gated into the register addressed by this field.
- If the Control Function bits transferred during phase 5 do not specify R ← 0/SIGN (see table 2–2), then CB3 specifies the source of the data gated onto the R-bus:
  - If CB3 is "0," the source is the output of the ALU.
  - If CB3 is "1," the data comes from an external source via the Data Multiplexer.
- If R ← 0/SIGN is specified, then CB3 specifies whether zero or the sign is propagated throughout the more significant byte.
- If the R-bus address is zero and the push-pull control bit was active (during phase 1), a data word is pushed onto the LIFO Stack from the R-bus.

2.4.2 MISCELLANEOUS TIMING SIGNALS

The control of the Flag Flip-Flops is indicated on the timing diagram (figure 2–4). A unique flag address is established during each machine cycle; at T2 the flag may be set, and/or at T6 the flag may be reset. It is thus possible to set, reset, or pulse a flag during a single machine cycle.

The Conditional Jump Multiplexer shares the same address lines as the Flag Flip-Flops. If a conditional jump is being performed, the condition is tested during T2.

2.4.3 DATA TRANSFER TIMING

Data transfers between the RALU and the Data Bus may occur at three times during each microcycle:

- During T4, the data word presently on the A-bus is gated onto the Data Bus. The information that appears on the Data Bus at this time either is output data destined for memory or an external device, or is an address value used for loading the Address Register latch (shared by memory and external devices).
- At T2, the contents of the R-bus (result of preceding microcycle) are gated onto the Data Bus. Primarily, the data provide inputs to the Conditional Jump Multiplexer to permit testing of the result of the operation performed on the previous microcycle.
- During T7, data to be transferred to the RALU appear on the Data Bus. The data may then be gated onto the R-bus by the RALU's Input/Output Multiplexer and, subsequently, may be stored in one of the working registers (AC0 through AC3).

2.5 IMP-16C OPERATION

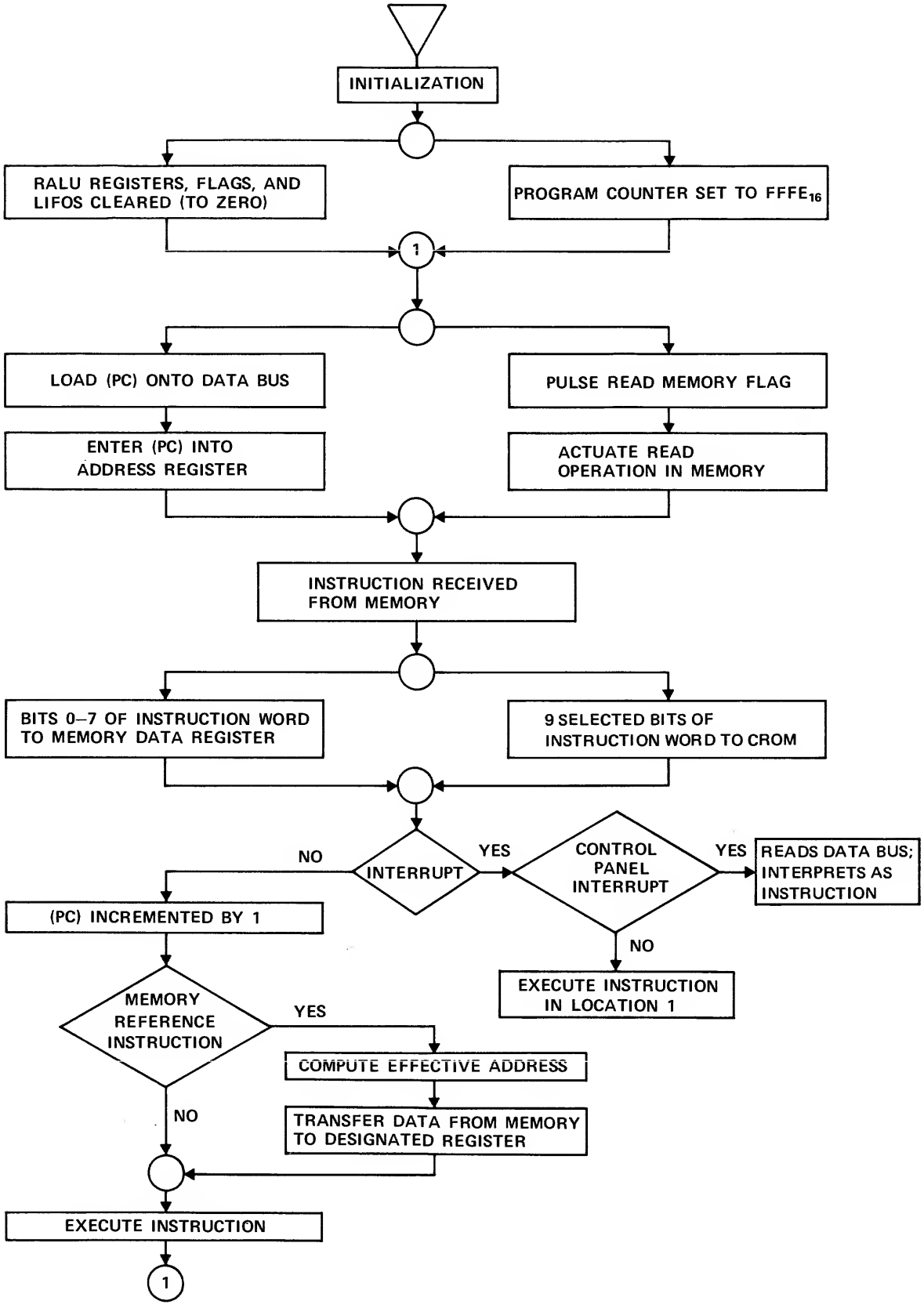
A flowchart of the events that occur during a typical operation of the IMP-16C is given in figure 2–5. The various events are further elaborated upon in the following paragraphs.

2.5.1 INITIALIZATION

When power is first applied to the processor, all RALU registers, flags, and the stack are cleared to zero. The microprogram then enters an initialization sequence, in which the Program Counter (PC) is set to a starting value of $FFFE_{16}$ (the next-to-last location in the top page of memory, assuming a maximum of 2^{16} memory locations). The reasons for choosing this location over location 0000_{16} or any other location can be explained as follows. In most applications, the first few executable statements in a macroprogram are usually initialize routines and other program segments of a supervisory nature. By keeping these in the top portion of memory, the supervisor programs can be stored in ROM for nonvolatility. Since the base page is directly accessible from anywhere in memory, it is desirable for it to be implemented with read/write memory.

2.5.2 INSTRUCTION FETCH

Following the initialize sequence, control is transferred to the first step in the fetch microroutine. During this first microcycle of the fetch routine, the contents of the Program Counter (PC) are sent out on the data bus at T4; at the same time, the Read Memory Flag is set high. This causes the external Address Register to be loaded with the contents of the PC. The Read Memory Flag actuates a read operation in the system memory. During T7 of the same microcycle, the 16-bit instruction comes back from memory ready to be decoded. At that time, bits 0 through 7 are placed in the RALU Memory Data Register, and bits 7 through 15 are loaded into the CROM. Bits 8 through 15 of the Memory Data Register are set equal to the value of bit 7.



NS00127

Figure 2–5. IMP-16C Instruction Execution Flowchart

In the next microcycle, the Program Counter is incremented to point to the next consecutive memory location.

A decision is made as to whether the instruction just read in needs to make reference to memory. If it does not (for register-register operations and other nonmemory reference instructions), then the instruction decode circuit steers control to a microprogram entry point that corresponds to the particular instruction. If the instruction requires a memory reference, then two additional microprogram steps are required to compute an effective address and to bring in the new word: first, the memory address is computed, and, second, the data word is transferred from memory to the register designated by the preceding instruction.

2.5.3 COMMUNICATIONS WITH MEMORY

During a memory-read operation, the microprogram sends an address on the data bus at T4; this address is loaded into the Address Register under control of the Read Memory Flag. Similarly, during a write-memory operation, the address is loaded under control of the Load Address Flag. These flags are further discussed in chapter 3.

When executing a memory-read operation, the processor sends out an address on the bus at T4 and expects data back at T7 of the same microcycle. A circuit, described in chapter 4, is included on the IMP-16C card to extend the interval between T4 and T7 to accommodate memories whose access times are longer than the normal T4-to-T7 interval.

A memory-write operation takes 2 microcycles, because both address and data have to be sent out. The address is sent out during T4 of the first microcycle and is latched in the address register. The output data destined for memory arrives during T4 of the next microcycle and can be used directly to write memory. In the IMP-16C, T4 is stretched for two additional periods to accommodate both the read and write delays necessary to communicate with the system memory.

2.5.4 EXECUTION OF INSTRUCTIONS

The CROM in the IMP-16C contains a microprogram that implements the standard instruction set. Each macroinstruction in a user's program is brought into the processor under control of the CROM's microprogram instruction fetch routine. This instruction is then decoded, and the ROM Address Control in the CROM directs the control sequence to an entry point in the microprogram. The sequence continues until execution is completed. Then, the CROM goes through another instruction-fetch cycle to bring in the next macroinstruction. The process is repeated continuously until directed otherwise by a macroinstruction such as Halt or an interrupt condition.

A test for an interrupt condition is made at the beginning of each instruction fetch. If the interrupt line is high, the CROM transfers control to a microprogram sequence that determines the type of interrupt that occurred. If a control panel interrupt is not active, the CROM assumes that a general interrupt is in effect. For a general interrupt, the contents of the PC are saved on the stack, and then the Interrupt Routine in memory location 1 is executed; otherwise, the Control Panel Subroutine is executed.

Chapter 3
IMP-16C INSTRUCTION SET

3.1 INTRODUCTION

Eight functional types of instructions comprise the basic IMP-16C instruction set:

- Load and Store
- Arithmetic
- Logical
- Skip
- Shifts
- Transfer of Control
- Register
- Input/Output and Miscellaneous

The instructions for each functional type are described as a group. For each instruction, the name of the instruction, its mnemonic, its word format, its operation in the form of an equation, and a succinct explanation of its operation are given. A tabulated summary of each type of instruction precedes the detailed descriptions. An extended instruction set (available as an option) is described in 3.7.

Before describing the instructions, brief descriptions of the registers referred to in the instruction descriptions are given.

The IMP-16C instructions and their assembler language opcode mnemonics are summarized in appendix A.

3.2 ARITHMETIC AND LOGIC UNITS REFERENCED IN IMP-16C INSTRUCTIONS

The units referenced in the ensuing description of the IMP-16C instructions are listed below and are shown in block diagram form in figure 3–1.

- Last-In/First-Out Stack (LIFOS)
- Register and Arithmetic Logic Unit (RALU) Flags
- Program Counter (PC)
- Accumulator 0 (AC0)
- Accumulator 1 (AC1)
- Accumulator 2 (AC2)
- Accumulator 3 (AC3)

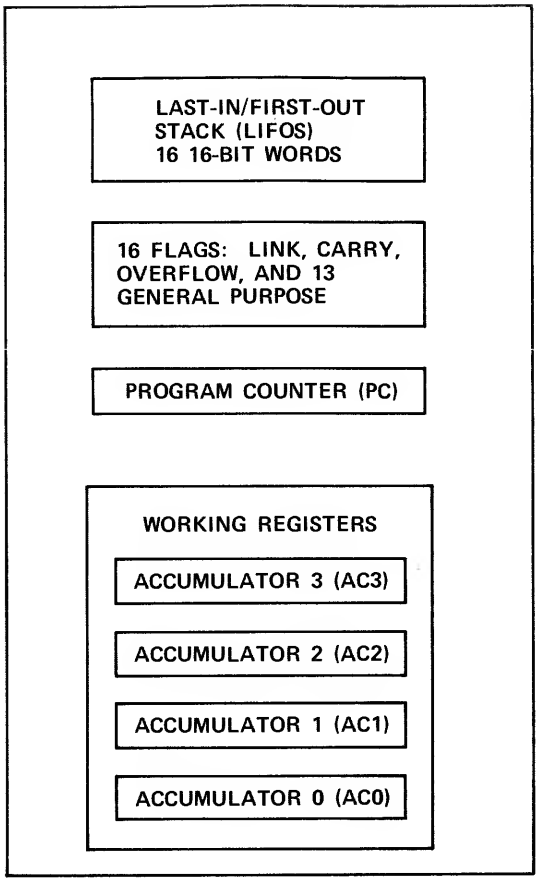


Figure 3–1. Arithmetic and Logic Units Referenced in IMP-16C Instructions

3.2.1 LAST-IN/FIRST-OUT STACK (LIFOS)

The IMP-16C has a hardware stack that data may be stored in or retrieved from on a last-in/first-out basis. The stack is 16 words deep and is accessible through the top location. As a data word is entered into the stack, the contents of the top location and each other location are pushed downward to the next lower level; if the stack is full, the word in the bottom location is lost. Conversely, the contents of the top location are pulled from the stack during retrieval of a data word; the top location and each lower location are replaced by the contents of the next lower location, and zeros are entered into the bottom location.

The stack is used primarily for saving status during interrupts and for saving subroutine return addresses. It may be used also for temporary storage of data using the PUSH, PULL, XCHRS, PUSHF, and PULLF instructions (described later in this chapter).

3.2.2 REGISTER AND ARITHMETIC LOGIC UNIT (RALU) FLAGS

There are 16 RALU status flags. These flags may be pushed onto the stack (saved) or may be loaded from the stack (restored). During such operations, the flags are configured as a 16-bit word: the L (Link), CY (Carry), and OV (Overflow) flags are the first, second, and third most significant bits, respectively, and the remaining 13 general-purpose flags comprise the remaining 13 less significant bits (figure 3–11).

Flags 0 and 12 are externally available. The L flag is primarily used in some shifting operations, and the CY and OV flags are adjuncts for arithmetic operations. The specific uses of the flags are elaborated upon in the appropriate instruction descriptions.

3.2.3 PROGRAM COUNTER (PC)

The Program Counter (PC) holds the address of the next instruction to be executed. When there is a branch to another address in the main memory, the branch address is set into the Program Counter. A skip instruction merely increments the Program Counter by 1, thus causing the one instruction to be skipped.

3.2.4 ACCUMULATORS 0, 1, 2, AND 3 (AC0, AC1, AC2, AND AC3)

The accumulators are used as working registers for data manipulation. Data may be fetched from a location in memory to an accumulator, and may be stored from an accumulator to a location in memory. The particular accumulator to take part in an operation is specified by the programmer in the appropriate instruction.

3.3 DATA AND INSTRUCTIONS

3.3.1 DATA REPRESENTATION

Data are represented in the IMP-16C in twos-complement integer notation. In this system, the negative of a number is formed by complementing each bit in the data word and adding 1 to the complemented number. The sign is indicated by the most significant bit. In the 16-bit word of the IMP-16C, when bit 15 is a "0," it denotes a positive number; when it is a "1," it denotes a negative number. Maximum number ranges for this system are $7FFF_{16}$ ($+32767_{10}$) and 8000_{16} (-32768_{10}). The carry flag is set to the value of the most significant bit position (bit 15) resulting from an add operation. The overflow flag is set to "1" if two numbers having like signs are added and the sign of the resulting sum is different from that of the operands.

3.3.2 INSTRUCTIONS

There are eight classes of IMP-16C instructions. Each class of instruction and the associated instructions are summarized in a table preceding the descriptions of the instructions. Also, the applicable instruction word formats are defined.

3.4 MEMORY ADDRESSING

The IMP-16C instruction set provides for direct and indirect memory addressing. For direct addressing, three distinct modes are available: base page (or absolute), program-counter relative, and indexed. The mode of addressing is specified by the XR field of the simplified instruction word format shown in figure 3–2.

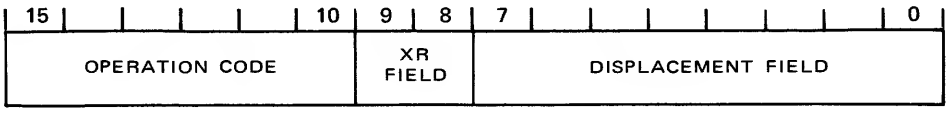


Figure 3–2. Instruction Word for Addressing Memory

3.4.1 BASE PAGE ADDRESSING

When the XR field is 00, it specifies base page addressing. Base page is directly accessible from any location in the address space of the memory. In this mode, the effective address is formed by setting bits 8 through 15 to zero and using the value of the 8-bit displacement field as an absolute address. Up to 256 words (locations 0 through 255) may be addressed in this way.

3.4.2 PROGRAM-COUNTER RELATIVE ADDRESSING

Program-counter relative addressing is specified when the XR field is 01. The displacement is treated as a signed number such that its sign bit (bit 7) is propagated to bits 8 through 15, and the effective address is formed by adding the contents of the PC to the resulting number. This permits PC-relative addressing –128 and +127 locations from the PC value; however, at the time of formation of the address, the PC has already been incremented in the microprogram and is pointing to the next macroinstruction. Because of this, the actual addressing range is from –127 to +128 from the current instruction.

3.4.3 INDEXED ADDRESSING

Indexed addressing is done with reference to Accumulator 2 or 3 (AC2 or AC3). In this mode, the displacement field is again interpreted as a signed 8-bit number from –128 to +127 with the sign (bit 7) extended through bits 8 through 15. The contents of the chosen index register (AC2 when xr = 10_2 and AC3 when xr = 11_2) are added to the number formed from the displacement value to yield an effective address that can reach any location in 65,536 words of memory.

Table 3–1. Summary of Addressing Modes

XR Field	Addressing Mode	Effective Address	Range
00	Base	EA = disp	$0 \leqslant \text{disp} \leqslant 255$
01	Relative to Program Counter	EA = disp + (PC)	$-128 \leqslant \text{disp} \leqslant 127$
10	Relative to AC2	EA = disp + (AC2)	$-128 \leqslant \text{disp} \leqslant 127$
11	Relative to AC3	EA = disp + (AC3)	$-128 \leqslant \text{disp} \leqslant 127$

3.4.4 INDIRECT ADDRESSING

Indirect addressing is accomplished by first calculating the effective address (EA) using the same method used for direct addressing; the memory location at this address contains a number that is then used as the address of the operand. The following instructions use indirect addressing:

- Load Indirect (see table 3–3)
- Store Indirect (see table 3–3)
- Jump Indirect (see table 3–7)
- Jump to Subroutine Indirect (see table 3–7)

3.5 NOTATION AND SYMBOLS USED IN IMP-16C INSTRUCTION DESCRIPTIONS

Refer to table 3–2 for definitions of the notation and symbols used in the IMP-16C instruction descriptions. The notations are given first in alphabetical order followed by the symbols. Upper-case mnemonics refer to fields in the instruction word; lower-case mnemonics refer to the numerical value of the corresponding fields. In cases where both lower- and upper-case mnemonics are composed of the same letters, only the lower-case mnemonic is given in table 3–2. The use of lower-case notation designates variables.

Table 3–2. Notation Used in Instruction Descriptions

Notation	Meaning
ACr	Denotes a specific working register (AC0, AC1, AC2, or AC3), where *r* is the number of the accumulator referenced in the instruction.
AR	Denotes the address register used for addressing memory or peripheral devices.
cc	Denotes the 4-bit condition code value for conditional branch instructions.
ctl	Denotes the 7-bit control-field value for flag, input/output, and miscellaneous instructions.
CY	Indicates that the Carry flag is set if there is a carry due to the instruction (either an addition or a subtraction).
disp	Stands for displacement value and it represents an operand in a nonmemory reference instruction or an address field in a memory reference instruction. It is an 8-bit, signed twos-complement number except when base page is referenced; in the latter case, it is unsigned.
dr	Denotes the number of a destination working register that is specified in the instruction-word field. The working register is limited to one of four: AC0, AC1, AC2, or AC3.
EA	Denotes the effective address specified by the instruction directly, indirectly, or by indexing. The contents of the effective address are used during execution of an instruction. See table 3–1.
fc	Denotes the number of the referenced flag (see table 3–13 under 3.6.8, Input/Output, Halt, and Flag Instructions).
INTEN	Denotes the Interrupt Enable control flag.
IOREG	Denotes an input/output register in a peripheral device.
L	Denotes 1-bit link (L) flag.
OV	Indicates that the overflow flag is set if there is an overflow due to the instruction (either an addition or a subtraction).
PC	Denotes the program counter. During address formation, it is incremented by 1 to contain an address 1 greater than that of the instruction being executed.
r	Denotes the number of a working register that is specified in the instruction-word field. The working register is limited to one of four: AC0, AC1, AC2, or AC3.
SEL	Denotes the Select control flag. It is used to select the carry or overflow for output on the carry and overflow (CYOV) line of the CPU, and to include the link bit (L) in shift operations.
sr	Denotes the number of a source working register that is specified in the instruction-word field. The working register is limited to one of four: AC0, AC1, AC2, or AC3.
xr	When not zero, this value designates the number of the register to be used in the indexed and relative memory-addressing modes. See table 3–1.

Table 3–2. Notation Used in Instruction Descriptions (Continued)

Notation	Meaning
( )	Denotes the contents of the item within the parentheses. (ACr) is read as "the contents of ACr." (EA) is read as "the contents of EA."
[ ]	Denotes "the result of."
~	Indicates the logical complement (ones complement) of the value on the right-hand side of ~.
→	Means "replaces."
←	Means "is replaced by."
@	Appearing in the operand field of an instruction, denotes indirect addressing.
∧	Denotes an AND operation.
∨	Denotes an OR operation.
∇	Denotes an exclusive OR operation.

3.6 INSTRUCTION DESCRIPTIONS

Each class and subclass of instruction is introduced by a table that lists and summarizes the instructions. The word format then is illustrated. Detailed descriptions are given, providing the following information:

- Name of instruction followed by operation code mnemonic in parentheses
- Operation Code in word format diagram
- Operation in equation notation
- Description of operation in detail

3.6.1 LOAD AND STORE INSTRUCTIONS

There are four instructions in this group. These are summarized in table 3–3 and then individually described. The word format is shown in figure 3–3.

Table 3–3. Load and Store Instructions

Instruction	OpCode	Operation	Assembler Format
LOAD	1000	(ACr) ← (EA)	LD r, disp(xr)
LOAD INDIRECT	1001	(ACr) ← ((EA))	LD r, @disp(xr)
STORE	1010	(EA) ← (ACr)	ST r, disp(xr)
STORE INDIRECT	1011	((EA)) ← (ACr)	ST r, @disp(xr)

NOTE

For indirect operations, the symbol @ must precede the memory location designated in the operand field of the assembler instruction.

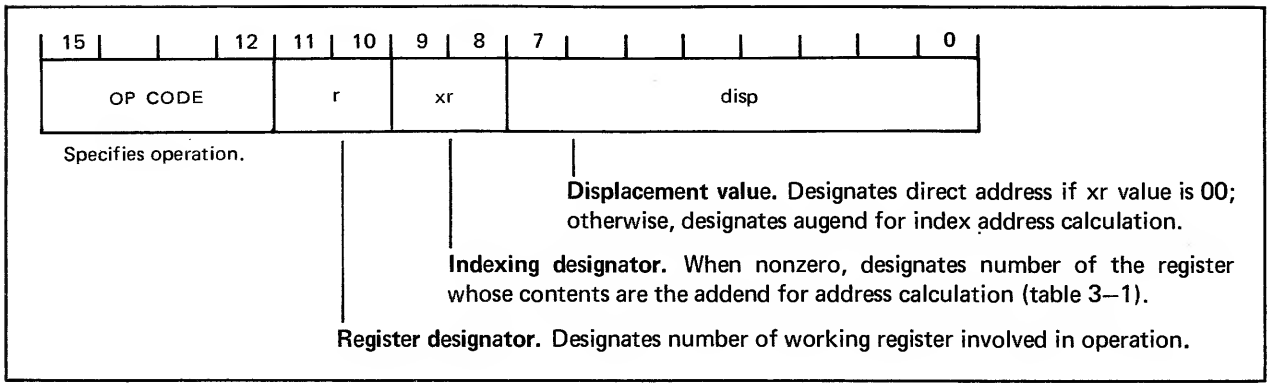
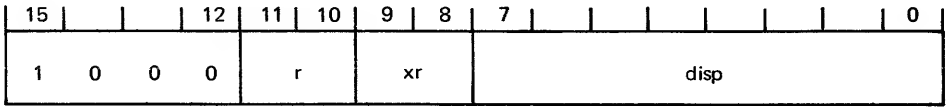


Figure 3–3. Load and Store Instruction Format

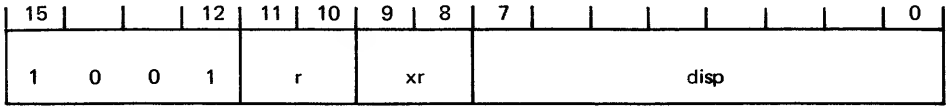
Load (LD)



Operation: (ACr) ← (EA)

Description: The contents of ACr are replaced by the contents of EA. The initial contents of ACr are lost; the contents of EA are unaltered.

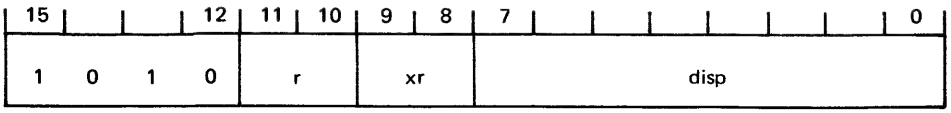
Load (LD) Indirect



Operation: (ACr) ← ((EA))

Description: The contents of ACr are replaced indirectly by the contents of EA. The initial contents of ACr are lost; the contents of EA and the location that designates EA are unaltered.

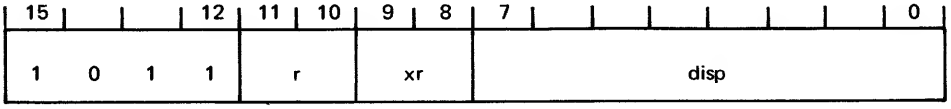
Store (ST)



Operation: (EA) ← (ACr)

Description: The contents of EA are replaced by the contents of ACr. The initial contents of EA are lost; the contents of ACr are unaltered.

Store (ST) Indirect



Operation: ((EA)) ← (ACr)

Description: The contents of EA are replaced indirectly by ACr. The initial contents of EA are lost; the contents of ACr and the location that designates EA are unaltered.

3.6.2 ARITHMETIC INSTRUCTIONS

There are two instructions in this group summarized in table 3–4 and then described individually. Either of these instructions may be carried out with any of the four general-purpose accumulators (AC0, 1, 2, or 3). The word format is shown in figure 3–4.

Table 3–4. Arithmetic Instructions

Instruction	OpCode	Operation	Assembler Format
ADD (ADD)	1100	(ACr) ← (ACr) + (EA),OV,CY	ADD r, disp(xr)
SUBTRACT (SUB)	1101	(ACr) ← (ACr) + ~ (EA) + 1,OV,CY	SUB r, disp(xr)

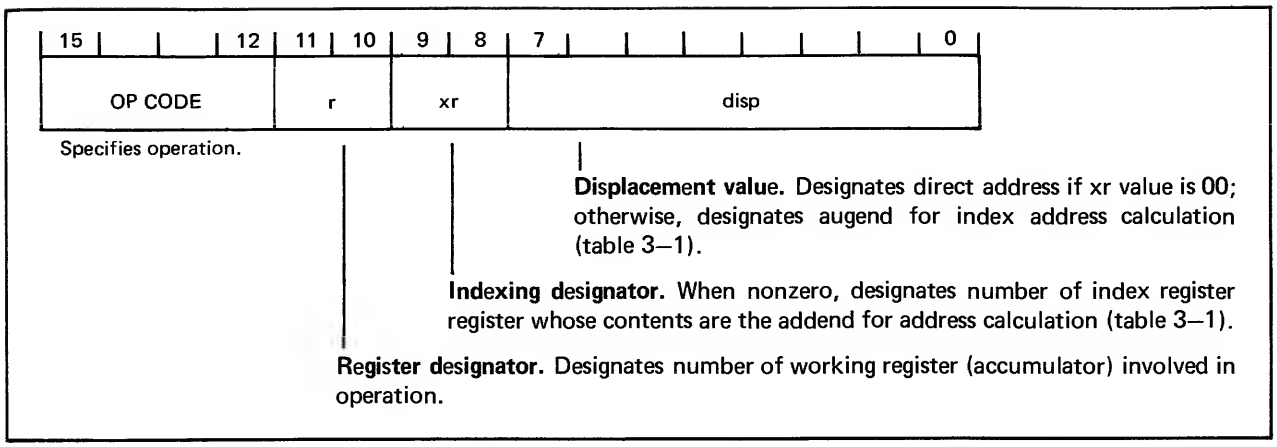
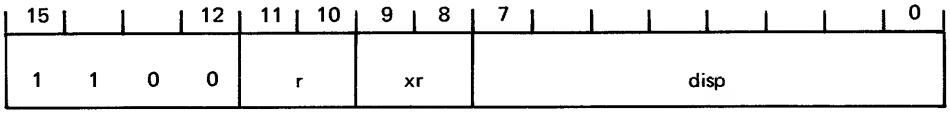


Figure 3–4. Arithmetic Instruction Format

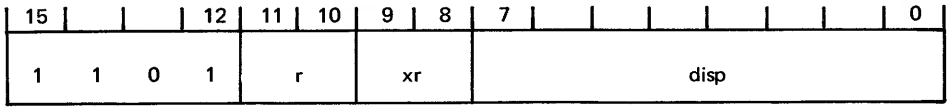
Add (ADD)



Operation: (ACr) ← (ACr) + (EA),OV,CY

Description: The contents of ACr are added algebraically to the contents of the effective memory location EA. The sum is stored in ACr, and the contents of EA are unaltered. The preceding contents of ACr are lost. The carry and overflow flags are set according to the result of the operation.

Subtract (SUB)



Operation: (ACr) ← (ACr) + ~ (EA) + 1,OV,CY

Description: The contents of ACr are added to the twos complement of the effective memory location EA. The result is stored in ACr, and the effective memory location is unaltered. The carry and overflow flags are set according to the result of the operation.

3.6.3 LOGICAL INSTRUCTIONS

There are two instructions in this group, summarized in table 3–5 and then described individually. Either of these instructions may be carried out with only two of the general-purpose accumulators, AC0 or AC1. The word format is shown in figure 3–5.

Table 3–5. Logical Instructions

Instruction	OpCode	Operation	Assembler Format
AND	01100	(ACr) ← (ACr) ∧ (EA)	AND r, disp(xr)
OR	01101	(ACr) ← (ACr) ∨ (EA)	OR r, disp(xr)

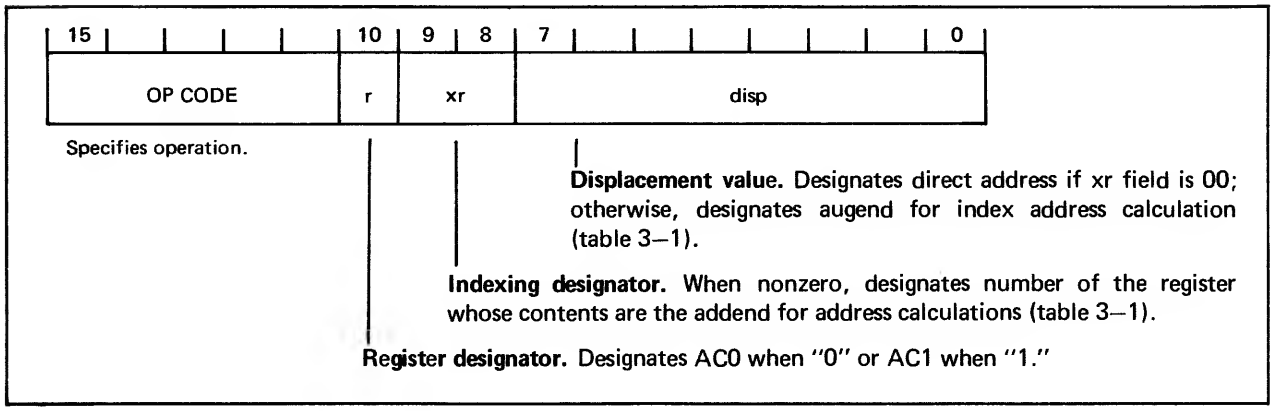
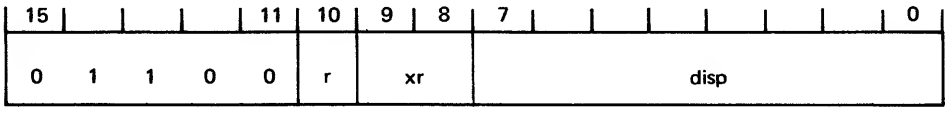


Figure 3–5. Logical Instruction Format

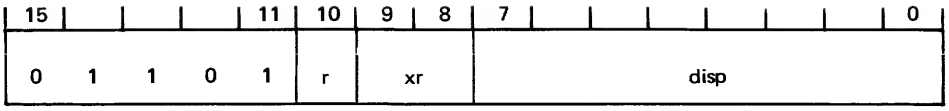
And (AND)



Operation: (ACr) ← (ACr) ∧ (EA)

Description: The contents of ACr (where r is either 0 or 1) and the contents of the effective memory location EA are ANDed, and the result is stored in ACr. The initial contents of ACr are lost, and the contents of EA are unaltered.

Or(OR)



Operation: (ACr) ← (ACr) ∨ (EA)

Description: The contents of ACr (where r is either 0 or 1) and the contents of the effective memory location EA are ORed inclusively, and the result is stored in ACr. The initial contents of ACr are lost, and the contents of EA are unaltered.

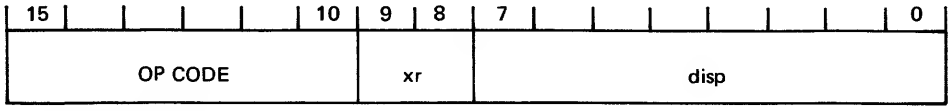
3.6.4 SKIP INSTRUCTIONS

Five instructions comprise the skip instructions, summarized in table 3–6. Three word formats are required and shown in figure 3–6.

Table 3–6. Skip Instructions

Instruction	Operation Code	Operation	Assembler Format
Memory References			
INCREMENT AND SKIP IF ZERO	011110	(EA) ← (EA) + 1; IF (EA) = 0, (PC) ← (PC) + 1	ISZ disp(xr)
DECREMENT AND SKIP IF ZERO	011111	(EA) ← (EA) – 1; IF (EA) = 0, (PC) ← (PC) + 1	DSZ disp(xr)
Register References			
SKIP IF GREATER	1110	IF (ACr) > (EA), (PC) ← (PC) + 1	SKG r, disp(xr)
SKIP IF NOT EQUAL	1111	IF (ACr) ≠ (EA), (PC) ← (PC) + 1	SKNE r, disp(xr)
Limited Register Reference			
SKIP IF AND IS ZERO	01110	IF [(ACr) ∧ (EA)] = 0, (PC) ← (PC) + 1	SKAZ r, disp(xr)

Memory Reference Skip Instruction

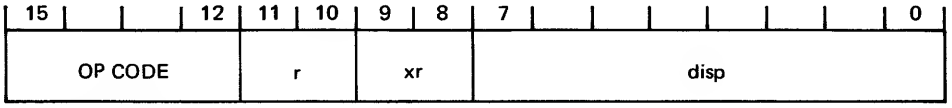


Specifies operation.

Displacement value. Designates direct address if xr value is 00; otherwise, designates augend for the address calculation (table 3–1).

Indexing designator. When nonzero, designates number of the register whose contents are the addend for address calculation (table 3–1).

Register Reference Instruction



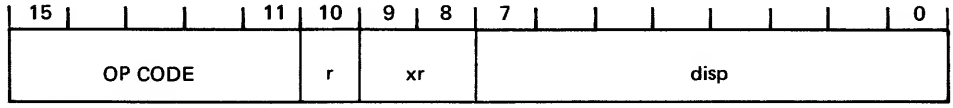
Specifies operation.

Displacement value. Designates direct address if xr value is 00; otherwise, designates augend for the address calculation (table 3–1).

Indexing designator. When nonzero, designates number of index register whose contents are the addend for address calculation (table 3–1).

Register designator. Specifies number of register whose contents are to be compared with contents of effective address in determining whether skip operation occurs. May be AC0, 1, 2, or 3.

Limited Register Reference Instruction



Specifies operation.

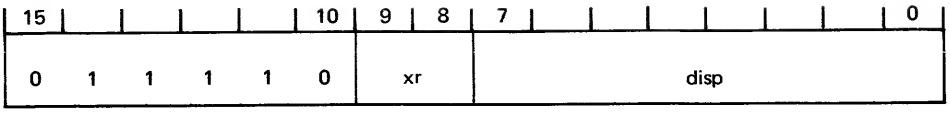
Displacement value. Designates direct address if xr value is 00; otherwise, designates augend for the address calculation (table 3–1).

Indexing designator. When nonzero, designates number of the register whose contents are the addend for address calculations (table 3–1).

Register designator. Specifies number of register whose contents are compared with contents of effective address in determining whether skip operation occurs. May be only AC0 or AC1.

Figure 3–6. Skip Instruction Formats

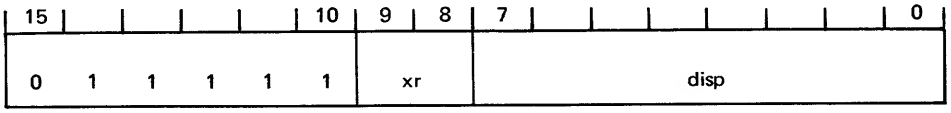
Increment and Skip If Zero (ISZ)



Operation: (EA) ← (EA) + 1; if (EA) = 0, (PC) ← (PC) + 1

Description: The contents of EA are incremented by 1. The new contents of EA are tested to determine whether they equal zero. If the new contents of EA equal zero, the contents of PC are incremented by 1, thus skipping the memory location designated by the initial contents of PC. The contents of EA are unaltered.

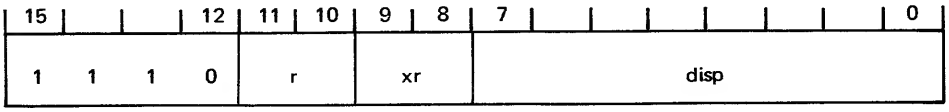
Decrement and Skip If Zero (DSZ)



Operation: (EA) ← (EA) – 1; if (EA) = 0, (PC) ← (PC) + 1

Description: The contents of EA are decremented by 1. The new contents of EA are tested to determine whether they equal zero. If the new contents of EA equal zero, the contents of PC are incremented by 1, thus skipping the memory location designated by the initial contents of PC.

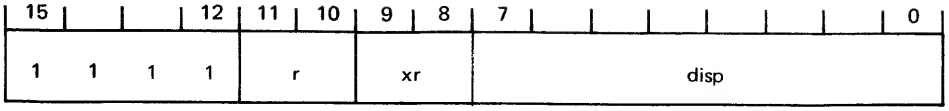
Skip If Greater (SKG)



Operation: If (ACr) > (EA), (PC) ← (PC) + 1

Description: The contents of ACr (when r is AC0, 1, 2, or 3) and the contents of the effective memory location EA are compared on an algebraic basis with due regard to the signs of the two operands. If the contents of ACr are greater than the contents of EA, the contents of PC are incremented by 1, thus skipping the memory location designated by the initial contents of PC. The initial contents of PC are lost. The contents of ACr and EA are unaltered.

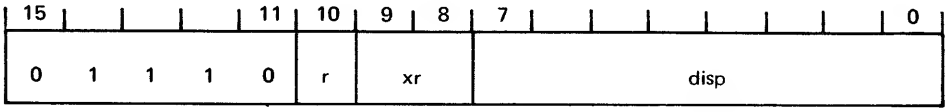
Skip If Not Equal (SKNE)



Operation: If ACr ≠ (EA), (PC) ← (PC) + 1

Description: The contents of ACr (where ACr is AC0, 1, 2, or 3) and the contents of the effective memory location EA are compared. If the contents of ACr and the effective memory location EA are not equal, the contents of PC are incremented, thus skipping the memory location designated by the initial contents of PC. The initial contents of PC are lost. The contents of ACr and EA are unaltered.

Skip If AND Is Zero (SKAZ)



Operation: If [(ACr) ∧ (EA)] = 0, (PC) ← (PC) + 1

Description: The contents of ACr (where r is either 0 or 1) and the contents of the effective memory location EA are ANDed. If the result equals zero, the contents of PC are incremented by 1, thus skipping the instruction designated by the initial contents of PC. The initial contents of PC are lost. The contents of ACr and EA are unaltered.

3.6.5 TRANSFER-OF-CONTROL INSTRUCTIONS

There are seven instructions in this group, summarized in table 3–7. Three word formats are required and shown in figure 3–7.

Table 3–7. Transfer-of-Control Instructions

Instruction	Operation Code	Operation	Assembler Format
Jumps			
JUMP	001000	(PC) ← EA	JMP disp(xr)
JUMP INDIRECT	001001	(PC) ← (EA)	JMP @disp(xr)
JUMP TO SUBROUTINE	001010	(STK) ← (PC); (PC) ← EA	JSR disp(xr)
JUMP TO SUBROUTINE INDIRECT	001011	(STK) ← (PC); (PC) ← (EA)	JSR @disp(xr)
Branch			
BRANCH-ON CONDITION	0001	IF CC IS TRUE (PC) ← (PC) + disp	BOC cc, disp

Table 3–7. Transfer-of-Control Instructions (Continued)

Instruction	Operation Code	Operation	Assembler Format
Returns			
RETURN FROM INTERRUPT	000000010	(PC) ← (STK) + ctl; INTEN FLAG SET	RTI ctl
RETURN FROM SUBROUTINE	000000100	(PC) ← (STK) + ctl	RTS ctl
JUMP TO SUBROUTINE IMPLIED	000000111	(STK) ← (PC); (PC) ← $FF80_{16}$ + ctl	JSRI ctl

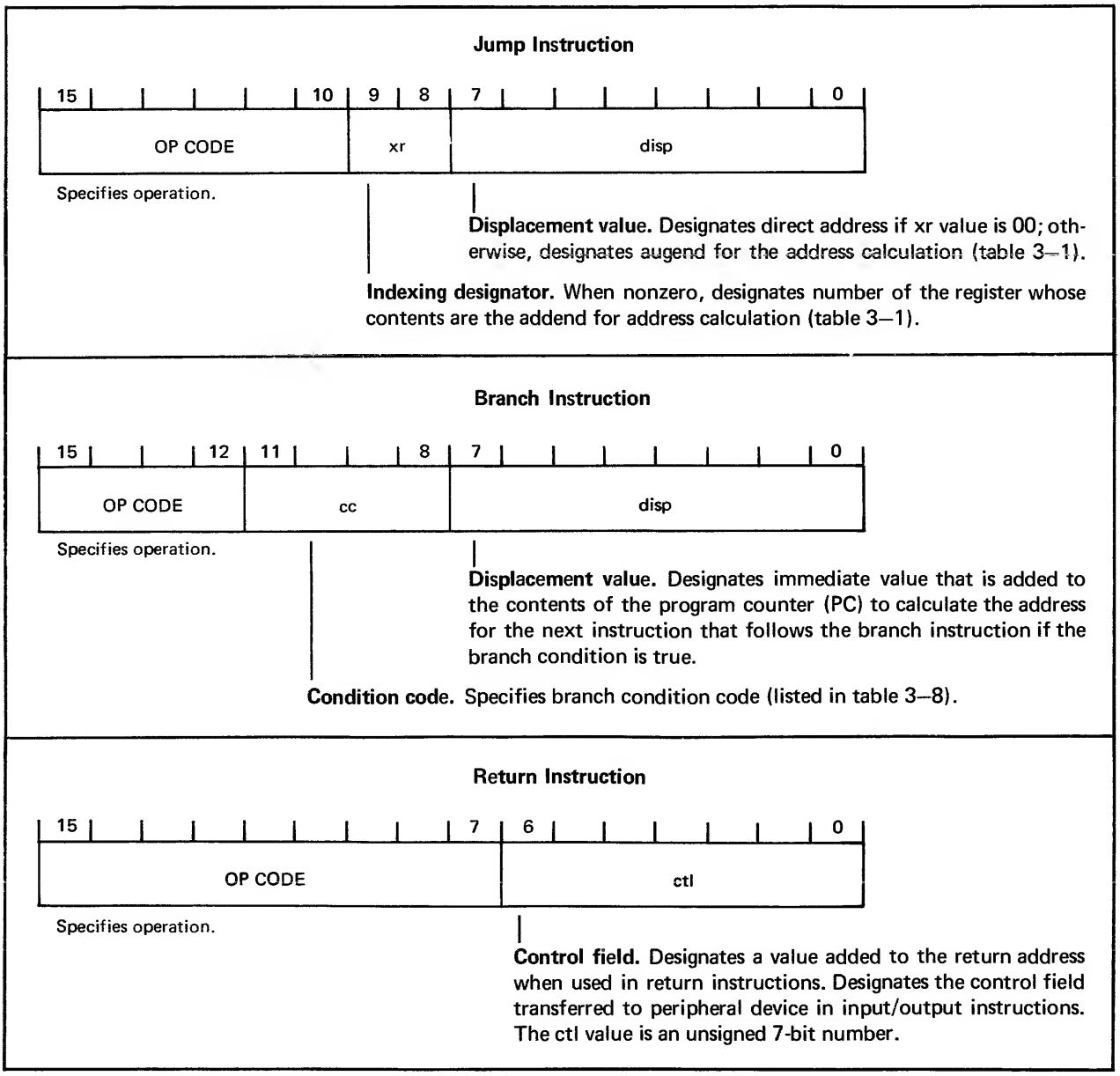
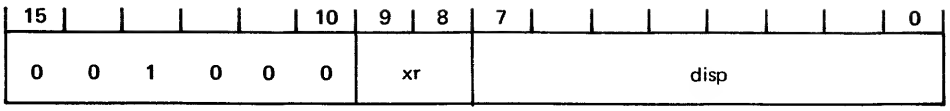


Figure 3–7. Transfer-of-Control Instruction Formats

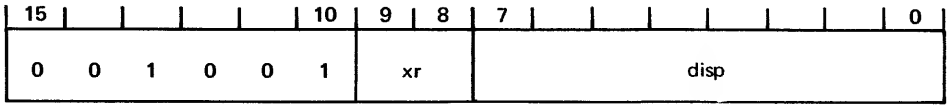
Jump(JMP)



Operation: (PC) ← EA

Description: The effective address EA replaces the contents of PC. The next instruction is fetched from the location designated by the new contents of PC.

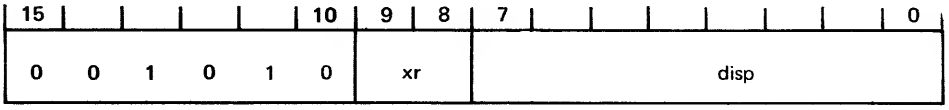
Jump (JMP) Indirect



Operation: (PC) ← (EA)

Description: The contents of the effective address (EA) replaces the contents of PC. The next instruction is fetched from the location designated by the new contents of PC.

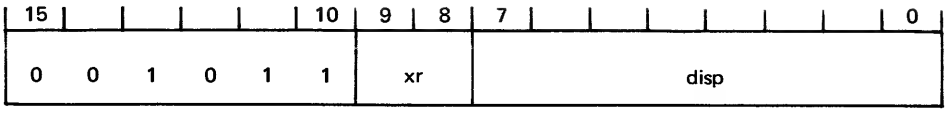
Jump to Subroutine (JSR)



Operation: (STK) ← (PC), (PC) ← (EA)

Description: The contents of PC are stored in the top of the stack. The effective address EA replaces the contents of PC. The next instruction is fetched from the location designated by the new contents of PC.

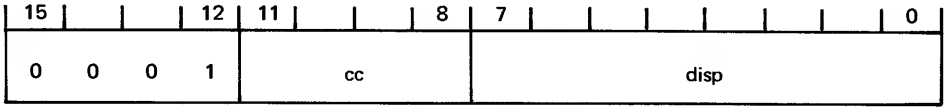
Jump to Subroutine (JSR) Indirect



Operation: (STK) ← (PC), (PC) ← (EA)

Description: The contents of PC are stored in the top of the stack. The contents of the effective address (EA) replace the contents of PC. The next instruction is fetched from the location designated by the new contents of PC.

Branch-On Condition (BOC)



Operation: (PC) ← (PC) + disp (sign extended from bit 7 through bit 15)

Description: There are 16 possible condition codes (cc). These are listed in table 3–8. If the condition for branching designated by cc is true, the value of disp (sign extended from bit 7 through bit 15) is added to the contents of PC, and the sum is stored in PC. The initial contents of PC are lost. Program control is transferred to the location specified by the new contents of PC.

NOTE

(1) PC is always incremented by 1 immediately following the fetching of an instruction, so the contents of PC during execution of an instruction is 1 greater than the address of that instruction. This must be considered during execution of the BOC instruction: for example, if the address of the BOC instruction is 100, then 101 is added to the value of disp (sign extended).

(2) The disp field is a signed 8-bit number, whose sign is extended from bit 7 through bit 15 to form a 16-bit number (including sign). Thus, the range of addressing with a BOC instruction is −127 to +128 relative to the address of the current instruction.

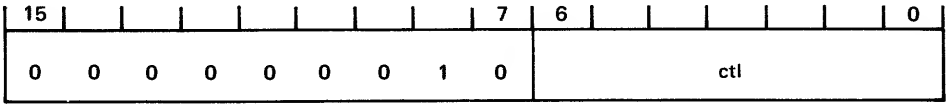
Table 3–8. Branch-On Condition Codes

Condition Code	Condition Tested	Remarks
0000	Interrupt Line = 1	Interrupt need not be tested by macroprogram
0001	(AC0) = 0	
0010	(AC0) $\geqslant$ 0	
0011	Bit 0 of AC0 = 1	
0100	Bit 1 of AC0 = 1	
0101	(AC0) $\neq$ 0	
0110	CONTROL PANEL INTERRUPT LINE = 1	
0111	CONTROL PANEL START = 1	
1000	STACK FULL LINE = 1	
1001	INTERRUPT ENABLE = 1	
1010	CARRY/OVERFLOW = 1	Carry if SEL = 0; overflow if SEL = 1
1011	(AC0) $\leqslant$ 0	
1100	User	Available for general-purpose use
1101	User	
1110	User	
1111	User	

NOTE

For both the following instructions (RTI and RTS), the ctl value is an unsigned 7-bit number.

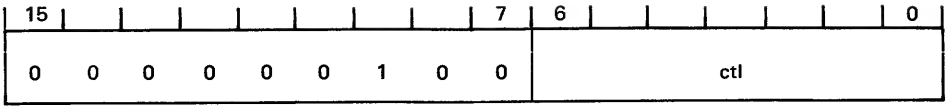
Return from Interrupt (RTI)



Operation: Set INT EN (interrupt enable flag)
(PC) ← (STK) + ctl

Description: The interrupt enable flag (INT EN) is set. The contents of PC are replaced by the sum of ctl and the contents of STK. Program control is transferred to the location specified by the new contents of PC. (RTI is used primarily to exit from an interrupt routine.)

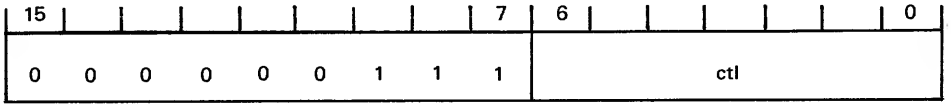
Return from Subroutine (RTS)



Operation: (PC) ← (STK) + ctl

Description: The contents of PC are replaced by the sum of ctl and the contents of STK. Program control is transferred to the location specified by the new contents of PC. (RTS is used primarily to return from subroutines entered by JSR.)

Jump to Subroutine Implied (JSRI)



Operation: (STK) ← (PC), (PC) ← $FF80_{16}$ + ctl

Description: The contents of PC are pushed onto the stack. The contents of PC are replaced by the address implied by the sum of the ctl value and the number $FF80_{16}$. This enables a subroutine jump to memory locations $FF80_{16}$ through $FFFF_{16}$ ($0 \leqslant ctl \leqslant 7F_{16}$).

3.6.6 SHIFT INSTRUCTIONS

Four instructions comprise this group. All four instructions may be used with the Link (L) bit by setting the SEL flag. This is accomplished with a Set Flag (SFLG) instruction before executing the shift or rotate instruction. Examples of shifting with and without SEL set are given in diagram form for each instruction in the descriptions that follow. Note that the SEL flag also affects the BOC instructions as indicated in table 3–8.

The shift instructions are summarized in table 3–9, and the word format is shown in figure 3–8.

All shift and rotate operations may be carried out with any of the four general-purpose accumulators, AC0, 1, 2, or 3.

Table 3–9. Shift Instructions

Instruction	Operation Code	Operation		Assembler Format
		SEL = 0	SEL = 1	
ROTATE LEFT (disp > 0)	010110	$(ACr_0) \leftarrow (ACr_{15})$, $(ACr_n) \leftarrow (ACr_{n-1})$	$(ACr_0) \leftarrow (L)$, $(L) \leftarrow (ACr_{15})$, $(ACr_n) \leftarrow (ACr_{n-1})$	ROL r, m
ROTATE RIGHT (disp < 0)	010110	$(ACr_{15}) \leftarrow (ACr_0)$, $(ACr_n) \leftarrow (ACr_{n+1})$	$(ACr_{15}) \leftarrow (L)$, $(L) \leftarrow (ACr_0)$, $(ACr_n) \leftarrow (ACr_{n+1})$	ROR r, m
SHIFT LEFT (disp > 0)	010111	$(ACr_n) \leftarrow (ACr_{n-1})$, $(ACr_0) \leftarrow 0$	$(L) \leftarrow (ACr_{15})$, $(ACr_n) \leftarrow (ACr_{n-1})$, $(ACr_0) \leftarrow 0$	SHL r, m
SHIFT RIGHT (disp < 0)	010111	$(ACr_{15}) \leftarrow 0$, $(ACr_n) \leftarrow (ACr_{n+1})$	$(ACr_{15}) \leftarrow (L)$, $(L) \leftarrow 0$, $(ACr_n) \leftarrow (ACr_{n+1})$	SHR r, m

NOTE: For all shift and rotate instructions, "m" denotes the number of positions to be shifted or rotated, and is equal to the absolute value of disp. See example 3 in appendix B.

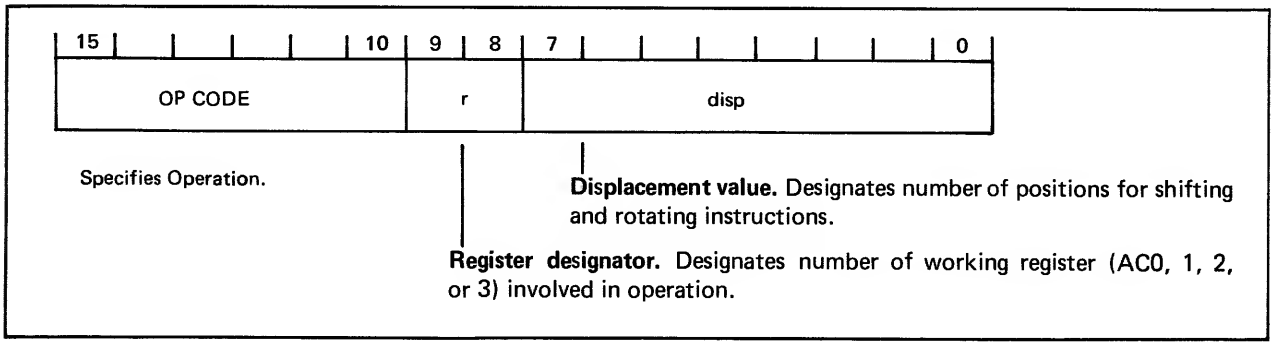
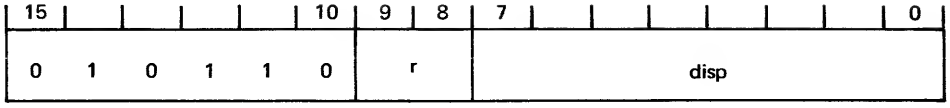


Figure 3 8. Shift Instruction Format

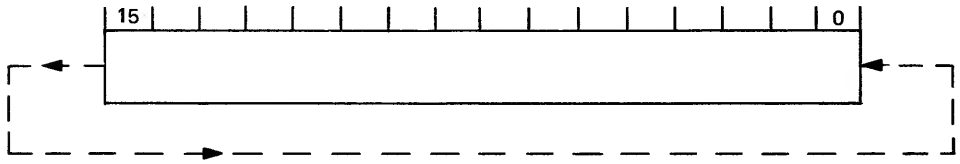
Rotate Left (ROL) *(for disp > 0)*



SEL = 0

Operation: $(ACr_0) \leftarrow (ACr_{15})$, $(ACr_n) \leftarrow (ACr_{n-1})$

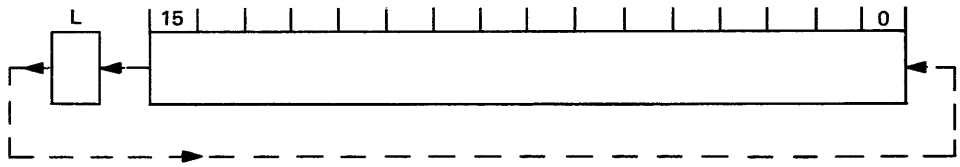
Description: The contents of ACr are shifted around to the left disp times. (ACr_{15}) replaces (ACr_0) for each shift.



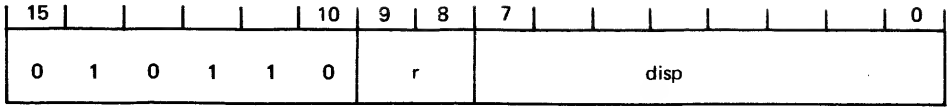
SEL = 1

Operation: $(ACr_0) \leftarrow (L)$, $(L) \leftarrow (ACr_{15})$, $(ACr_n) \leftarrow (ACr_{n-1})$

Description: The contents of ACr are shifted around to the left disp times. (L) replaces (ACr_0), and (ACr_{15}) replaces (L) for each shift.



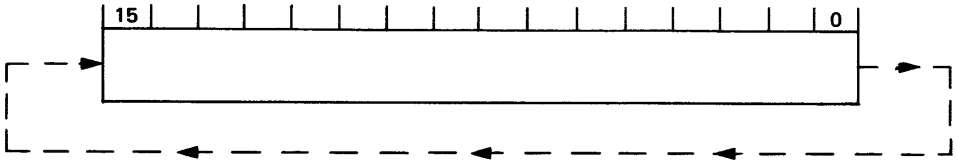
Rotate Right (ROR) *(for disp < 0)*



SEL = 0

Operation: $(ACr_{15}) \leftarrow (ACr_0)$, $(ACr_n) \leftarrow (ACr_{n+1})$

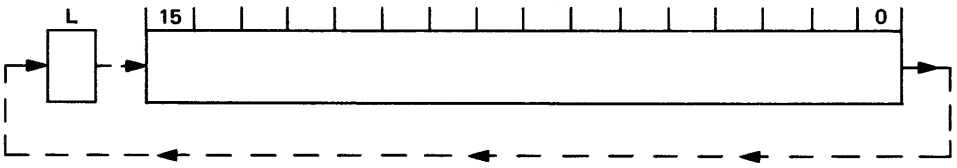
Description: The contents of ACr are shifted around to the right disp times. (ACr_0) replaces (ACr_{15}) for each shift.



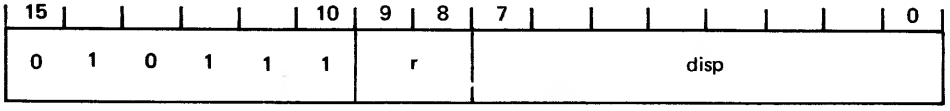
SEL = 1

Operation: $(ACr_{15}) \leftarrow (L)$, $(L) \leftarrow (ACr_0)$, $(ACr_n) \leftarrow (ACr_{n+1})$

Description: The contents of ACr are shifted around to the right disp times. (L) replaces (ACr_{15}), and (ACr_0) replaces (L) for each shift.



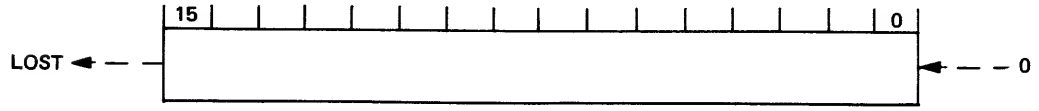
Shift Left (SHL) *(for disp > 0)*



SEL = 0

Operation: $(ACr_n) \leftarrow (ACr_{n-1})$, $(ACr_0) \leftarrow 0$

Description: The contents of ACr are shifted to the left disp times. (ACr_{15}) is lost, and zero replaces (ACr_0) for each shift.

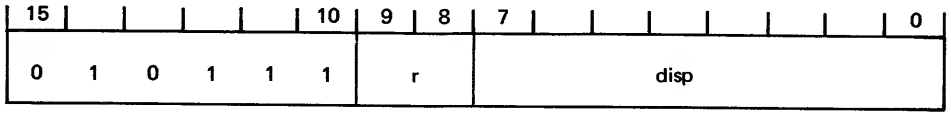


SEL = 1

Operation: $(L) \leftarrow (ACr_{15})$, $(ACr_n) \leftarrow (ACr_{n-1})$, $(ACr_0) \leftarrow 0$

Description: The contents of ACr are shifted to the left disp times. (L) is lost. (ACr_{15}) replaces (L), and zero replaces (ACr_0) for each shift.

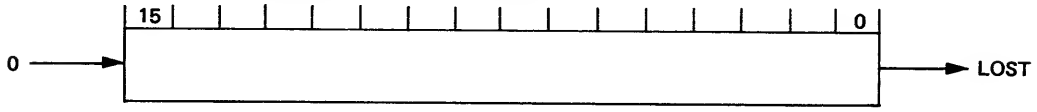
Shift Right (SHR) *(for disp < 0)*



SEL = 0

Operation: $(ACr_{15}) \leftarrow 0$, $(ACr_n) \leftarrow (ACr_{n+1})$

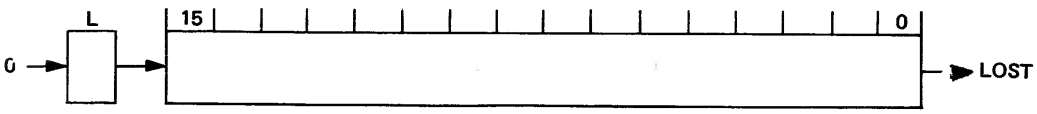
Description: The contents of ACr are shifted to the right disp times. (ACr_0) is lost, and zero replaces (ACr_{15}) for each shift.



SEL = 1

Operation: $(ACr_{15}) \leftarrow (L)$, $(L) \leftarrow 0$, $(ACr_n) \leftarrow (ACr_{n+1})$

Description: The contents of ACr are shifted to the right disp times. (ACr_0) is lost, (L) replaces ACr_{15}, and zero replaces (L) for each shift.



3.6.7 REGISTER INSTRUCTIONS

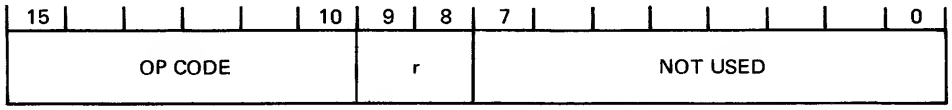
There are eleven instructions in this group, summarized in table 3–10. Three word formats are required and are shown in figure 3–9.

Table 3–10. Register Instructions

Instruction	Operation Code	Operation	Assembler Format
Register and Stack			
PUSH ONTO STACK	010000	(STK) ← (ACr)	PUSH r
PULL FROM STACK	010001	(ACr) ← (STK)	PULL r
EXCHANGE REGISTER AND STACK	010101	(STK) ← (ACr), (ACr) ← (STK)	XCHRS r
Register and Immediate			
LOAD IMMEDIATE	010011	(ACr) ← disp (sign extended)	LI r, disp
ADD IMMEDIATE, SKIP IF ZERO	010010	(ACr) ← (ACr) + disp (sign extended), OV,CY; if (ACr) = 0, (PC) ← (PC) + 1	AISZ r, disp
COMPLEMENT AND ADD IMMEDIATE	010100	(ACr) ← ~ (ACr) + disp (sign extended)	CAI r, disp

Instruction	Operation Code			Operation	Assembler Format
	OP1	OP2	OP3		
Register to Register					
REGISTER ADD	0011	0	00	(ACdr) ← (ACsr) + (ACdr), OV, CY	RADD sr, dr
REGISTER EXCHANGE	0011	1	00	(ACsr) ← (ACdr), (ACdr) ← (ACsr)	RXCH sr, dr
REGISTER COPY	0011	1	01	(ACdr) ← (ACsr)	RCPY sr, dr
REGISTER EXCLUSIVE-OR	0011	1	10	(ACdr) ← (ACsr) ∇ (ACdr)	RXOR sr, dr
REGISTER AND	0011	1	11	(ACdr) ← (ACsr) ∧ (ACdr)	RAND sr, dr

Register and Stack Instructions

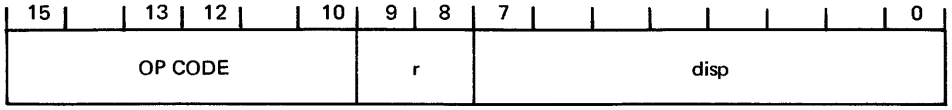


Specifies operation.

Not used. These bits are coded to zeros and are ignored for these instructions.

Register designator. Designates number of working register (AC0, 1, 2, or 3) involved in operation.

Register and Immediate Instructions

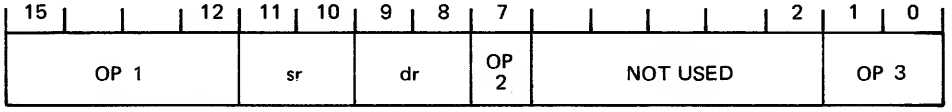


Specifies operation.

Displacement value. Designates immediate value, which is an operand in those instructions involving an immediate operand.

Register designator. Designates number of working register (AC0, 1, 2, or 3) involved in operation.

Register-to-Register Instructions



Specifies operation with bits 0, 1, and 7.

OpCode 3. Bits 0, 1, and 7 with bits 12 through 15 comprise the operation code and specify the operation.

Not used. These bits are coded to zeros and are ignored for this class.

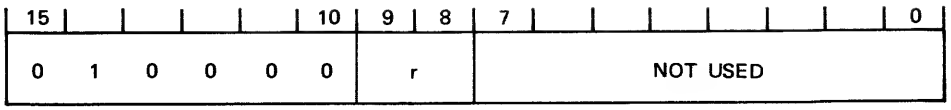
OpCode 2. Bits 0, 1, and 7 with bits 12 through 15 comprise the operation code and specify the operation.

Destination register designator. Designates number of one of the two operand registers involved in an operation. Result is placed in the destination register, whose original contents are lost.

Source register designator. The number of one of the two operand registers involved in an operation. Its contents are not altered by the operation except during RXCH (register exchange).

Figure 3–9. Register Instruction Formats

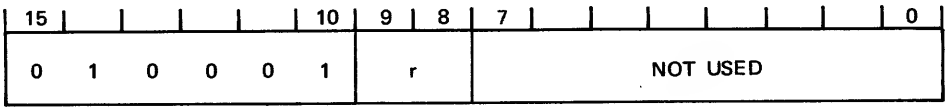
Push onto Stack (PUSH)



Operation: (STK) ← (ACr)

Description: The stack is pushed by the contents of the register (AC0, 1, 2, or 3) designated by r. Thus, the top of the stack then holds the contents of ACr, and the contents of all other levels in the stack are moved down one level. If the stack is full before the push occurs, the contents of the lowest level are lost. The initial contents of ACr are unaltered.

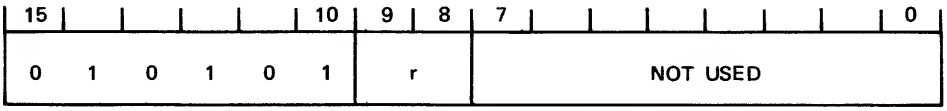
Pull from Stack (PULL)



Operation: (ACr) ← (STK)

Description: The stack is pulled. The contents from the top of the stack replace the contents of register number r (AC0, 1, 2, or 3). The initial contents of ACr are lost. The contents of each level of the stack moves up one level. Zeros enter the bottom of the stack.

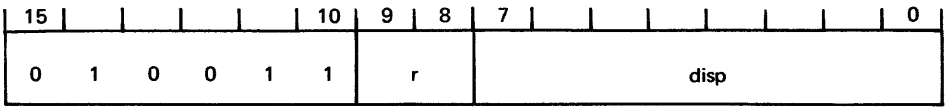
Exchange Register and Stack (XCHRS)



Operation: (STK) ← (ACr), (ACr) ← (STK)

Description: The contents of the top of the stack STK and the register designated by r (AC0, 1, 2, or 3) are exchanged.

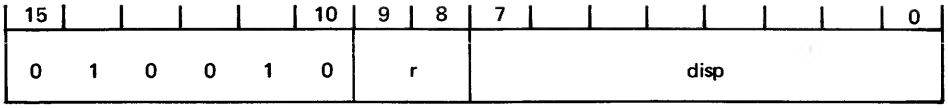
Load Immediate (LI)



Operation: (ACr) ← disp (sign extended)

Description: The value of disp with sign bit 7 extended through bit 15 replaces the contents of ACr (AC0, 1, 2, or 3). The initial contents of ACr are lost. The immediate operand range is –128 to +127.

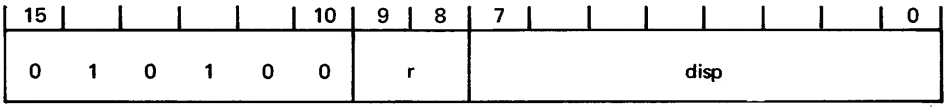
Add Immediate, Skip If Zero (AISZ)



Operation: (ACr) ← (ACr) + disp (sign extended), OV, CY
If new (ACr) = 0, (PC) ← (PC) + 1

Description: The contents of register ACr are replaced by the sum of the contents of ACr and disp (sign bit 7 extended through bit 15). The initial contents of ACr are lost. The overflow and carry flags are set according to the result of the operation. If the new contents of ACr equal zero, the contents of PC are incremented by 1, thus skipping the next memory location. The immediate operand range is -128 to +127.

Complement and Add Immediate (CAI)



Operation: (ACr) ← ~ (ACr) + disp (sign extended)

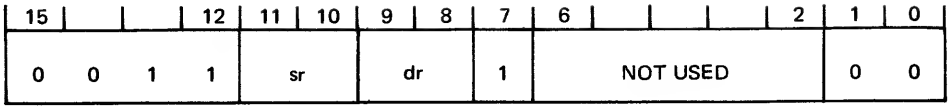
Description: The contents of register ACr are complemented and then added to disp (sign bit extended through bit 15). The result is then stored in ACr. The initial contents of ACr are lost. The immediate operand range is -128 to +127. Note that the carry and overflow flags are not affected by this instruction.

Register Add (RADD)

Operation: (ACdr) ← (ACsr) + (ACdr), OV, CY

Description: The contents of the destination register ACdr (AC0, 1, 2, or 3) are replaced by the sum of the contents of ACdr and the source register ACsr (AC0, 1, 2, or 3). The initial contents of ACdr are lost, and the contents of ACsr are unaltered. The overflow and carry flags are set according to the result of the operation.

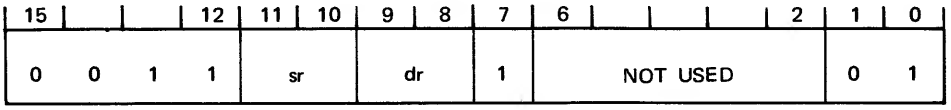
Register Exchange (RXCH)



Operation: (ACsr) ← (ACdr), (ACdr) ← (ACsr)

Description: The contents of ACsr (AC0, 1, 2, or 3) and ACdr (AC0, 1, 2, or 3) are exchanged.

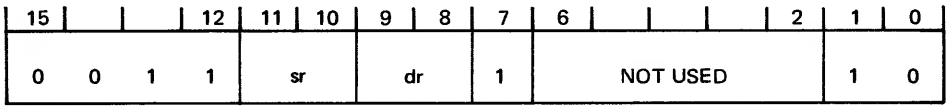
Register Copy (RCPY)



Operation: (ACdr) ← (ACsr)

Description: The contents of the destination register ACdr (AC0, 1, 2, or 3) are replaced by the contents of the source register ACsr (AC0, 1, 2, or 3). The initial contents of ACdr are lost, and the initial contents of ACsr are unaltered.

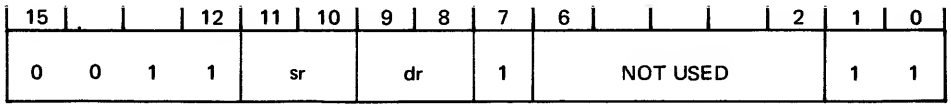
Register Exclusive Or (RXOR)



Operation: (ACdr) ← (ACdr) ∇ (ACsr)

Description: The contents of the destination register ACdr (AC0, 1, 2, or 3) are replaced by the result of exclusively ORing the contents of ACdr with the contents of the source register ACsr (AC0, 1, 2, or 3). The initial contents of ACdr are lost, and the initial contents of ACsr are unaltered.

Register And (RAND)



Operation: (ACdr) ← (ACdr) ∧ (ACsr)

Description: The contents of the destination register ACdr (AC0, 1, 2, or 3) are replaced by the result of ANDing the contents of ACdr with the contents of the source register ACsr (AC0, 1, 2, or 3). The initial contents of ACdr are lost, and the initial contents of ACsr are unaltered.

3.6.8 INPUT/OUTPUT, HALT, AND FLAG INSTRUCTIONS

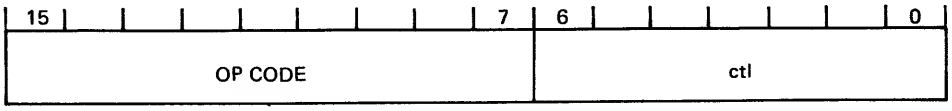
Seven instructions comprise this group, summarized in table 3–11. Three word formats are required and are shown in figure 3–10.

Table 3–11. Input/Output, Halt, and Flag Instructions

Instruction	Operation Code	Operation	Assembler Format
Input/Output			
REGISTER IN	000001000	(AR) ← ctl + (AC3); (AC0) ← (IOREG)	RIN ctl
REGISTER OUT	000001100	(AR) ← ctl + (AC3); (IOREG) ← (AC0)	ROUT ctl
Halt			
HALT	000000000	Processor halts.	HALT
Status Flags			
PUSH STATUS FLAGS ONTO STACK	000000001	(STK) ← (STATUS FLAGS)	PUSHF
PULL STATUS FLAGS FROM STACK	000000101	(STATUS FLAGS) ← (STK)	PULLF

Instruction	Operation Code		Operation	Assembler Format
	OP1	OP2		
Control Flags				
SET FLAG	00001	0	fc set; (AR) ← ctl	SFLG fc
PULSE FLAG	00001	1	fc pulsed; (AR) ← ctl	PFLG fc

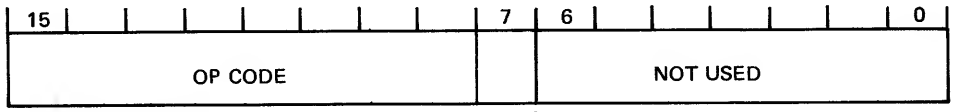
Input/Output Instructions



Specifies operation.

Control field. The value of ctl represents control data for input/output operations.

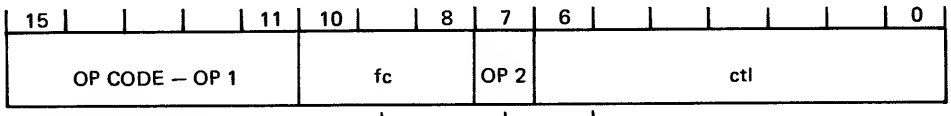
Status Flag and Halt Instructions



Specifies operation.

Not used. These bits are coded to zero and are ignored for these instructions.

Control Flag Instructions



Specified operation with bit 7.

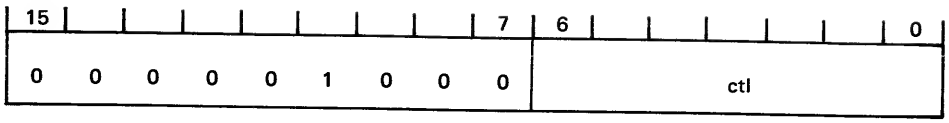
Control field. Designates a control value that is transferred to AR (address register) for control flag instructions. Bit 7 of the particular instruction, a 0 for SFLG and a 1 for PFLG, is extended through bit 15, and this extended field is transferred to bits 7 through 15 of AR.

OpCode 2. Bit 7 with bits 11 through 15 comprise the operation code and specify the operation.

Flag code. Specifies one of eight flags that may be set or pulsed. Flag codes are listed in table 3–13.

Figure 3–10. Input/Output, Halt, and Flag Instruction Formats

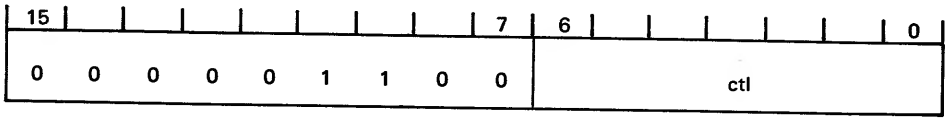
Register In (RIN)



Operation: (AR) ← ctl + (AC3), (AC0) ← (IOREG)

Description: The contents of AR (address register) are replaced by the sum of ctl and the contents of AC3. The new contents of AR constitute the address of a peripheral device and a command, both of which are received by the addressed peripheral device. The peripheral device responds by transferring the contents of its input/output register (IOREG) to the processor AC0.

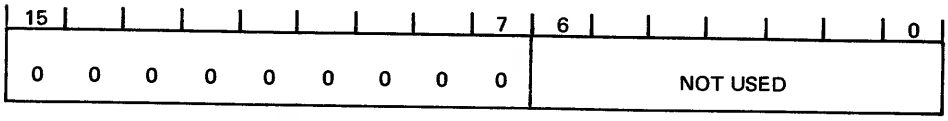
Register Out (ROUT)



Operation: (AR) ← ctl + (AC3), (IOREG) ← (AC0)

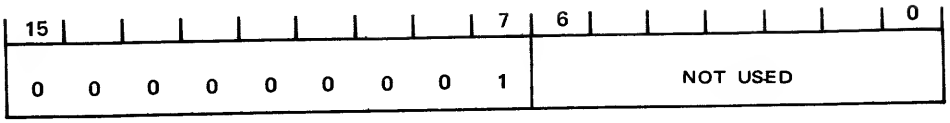
Description: The contents of AR (address register) are replaced by the sum of ctl and the contents of AC3. The new contents of AR constitute the address of a peripheral device and a command, both of which are received by the addressed peripheral device. The processor then transfers the contents of AC0 to IOREG in the peripheral device.

Halt (HALT)



Description: The processor halts and remains halted until the START input makes a transition from logic "1" to "0." A switch may be wired to this input such that a logic "1" is applied momentarily, and then released to a logic "0" level, thereby providing the necessary transition. The HALT flag is set by this instruction, and it remains set until the processor comes out of the halted condition.

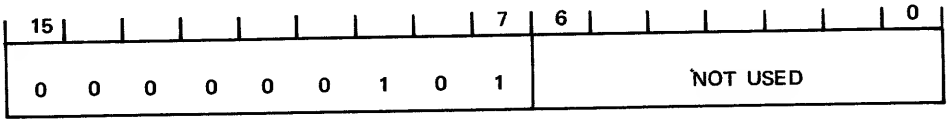
Push Status Flags onto Stack (PUSHF)



Operation: (STK) ← (STATUS FLAGS)

Description: The contents of the top of the stack STK are replaced by the contents of the status flags. See figure 3–11 for the configuration of processor flags on the stack and table 3–12 for processor flag definitions. The previous contents of the top of the stack and lower levels are pushed down one level. The contents of the lowest level of the stack are lost.

Pull Status Flags from Stack (PULLF)



Operation: (STATUS FLAGS) ← (STK)

Description: The contents of the status flags are replaced by the contents of the top of the stack (STK). See figure 3–11 for the configuration of processor flags on the stack and table 3–12 for processor-flag definitions. The previous contents of lower levels of the stack are pulled up by one level with zeros replacing the contents of the lowest level.

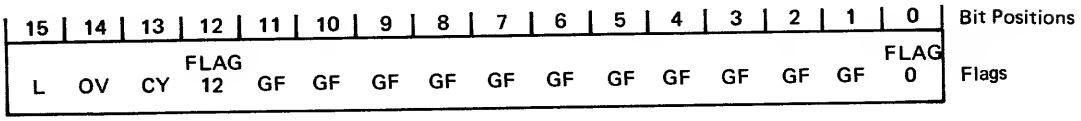


Figure 3–11. Configuration of Status Flags

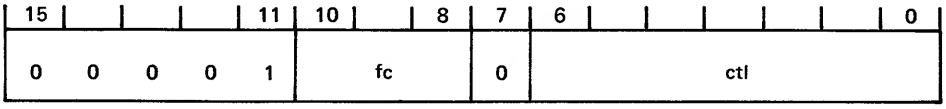
NOTE

Status flags 0 and 12 are externally available.

Table 3–12. Status Flags

Bit Position	Flag Name	Mnemonic	Significance
15	Link	L	Used for double-word shifts
14	Overflow	OV	Set if an arithmetic overflow occurs
13	Carry	CY	Set if a carry occurs (from most significant bit) during an arithmetic operation
12 through 0	General-Purpose Flags	GF	Use specified by programmer

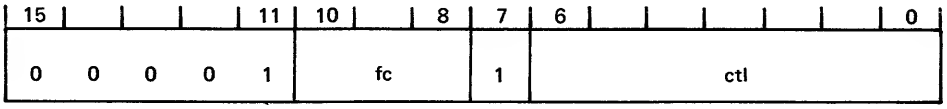
Set Flag (SFLG)



Operation: FC set, (AR) ← ctl (bit 7 extended through bit 15; that is, bits 8 through 15 set to 0)

Description: The control flag designated by the flag code FC is set. The contents of the address register AR are replaced by the value of ctl. Flag codes are defined in table 3–13.

Pulse Flag (PFLG)



Operation: FC pulsed, (AR) ← ctl (bit 7 extended through bit 15; that is, bits 8 through 15 set to 1)

Description: The control flag designated by the flag code FC is pulsed (note 2 below). The contents of the address register AR are replaced by the value of ctl. Flag codes are defined in table 3–13.

NOTE

(1) SFLG and PFLG refer to control flags external to the RALUs. These flags should not be confused with the RALU-internal flags, which are referenced by PUSHF and PULLF.

(2) Pulsing a control flag sets the flag at T2 and resets it at T6 during the same microcycle. The flag remains reset until again set or pulsed.

(3) The ctl value plus the extended bit 7 through bit 15 are transferred to the AR (address register). This word in the AR has no specified use and may be used as desired by the system programmer.

Table 3–13. Control Flag Codes

FC	Flag Mnemonic	Significance
000	F8	User Specified
001	INT EN	Interrupt Enable
010	SEL	Select
011	F11	User Specified
100	F12	User Specified
101	F13	User Specified
110	F14	User Specified
111	F15	User Specified

NOTE: The flag designated by the flag code (fc) is set or pulsed. Only control flags with addresses between 8 and 15 may be accessed with these instructions. (The flag address is 8, binary 1000, greater than the corresponding flag code, FC.) This is done because control flags with flag addresses 0 through 7 are used for various input/output operations controlled by the processor, and are not usable by the programmer. The SEL flag selects between CY and OV for output on the CYOV line; it also selects the LINK bit for inclusion in shift and rotate operations.

3.7 EXTENDED INSTRUCTION SET

An extended instruction set is available for the IMP-16C in the form of a second CROM. This set comprises 17 instructions divided into 5 categories:

- Double-word Arithmetic
- Load and Store Byte
- Bit Operations
- Interrupt Handling Operations
- Transfer of Control Operations

The instructions for each functional type are described as a group. For each instruction, the name of the instruction, its mnemonic, its word format, its operation in the form of an equation, and a succinct explanation of its operation are given. A tabulated summary of each type of instruction precedes the detailed descriptions.

3.7.1 DOUBLE-WORD MEMORY ADDRESSING

Six of the seventeen instructions use a double-word instruction format. These instructions use direct and indirect addressing exactly as described in 3.4, although the instruction format is different for these six memory-reference instructions from that described in 3.4. The six memory-reference instructions are as follows:

- MULTIPLY (MPY)
- DIVIDE (DIV)

- DOUBLE PRECISION ADD (DADD)
- DOUBLE PRECISION SUBTRACT (DSUB)
- LOAD BYTE (LDB)
- STORE BYTE (STB)

The modified format (shown in figure 3–12) is a double-word instruction format with a 16-bit displacement field. When using these instructions, all of memory is directly addressable, although all indexing modes may still be used. The addressing modes are unchanged from those indicated in table 3–1. It is important to note when considering PC relative addressing, that the PC contains the address of the displacement word of the instruction.

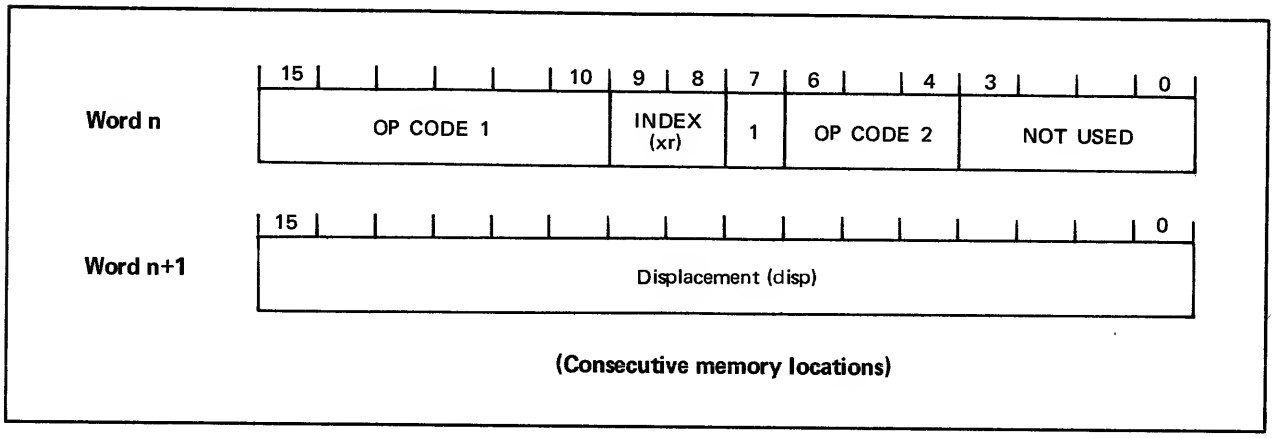


Figure 3–12. Double-Word Memory Reference Instruction Format

3.7.2 DOUBLE-WORD ARITHMETIC INSTRUCTIONS

There are four instructions in this group. These are summarized in table 3–14 and then individually described. The word format for the two double-precision instructions is shown in figure 3–13. The multiply and divide instructions have formats as shown in figure 3–13.

Table 3–14. Double-Word Arithmetic Instructions

Instruction	Operation Code		Operation	Assembler Format
	OP1	OP2		
DOUBLE PRECISION ADD	000001	1010	(AC0), (AC1) ← (AC0), (AC1) + (EA), (EA+1); OV; CY; SEL ← 0	DADD disp(xr)
DOUBLE PRECISION SUBTRACT	000001	1011	(AC0), (AC1) ← (AC0), (AC1) + ∿ (EA), ∿ (EA+1) + 1; OV; CY; SEL ← 0	DSUB disp(xr)
MULTIPLY (MPY) ((EA) ≥ 0)	000001	1000	(AC0), (AC1) ← (AC1)*(EA); SEL ← 0; L altered	MPY disp(xr)
DIVIDE (DIV) ((EA) > 0)	000001	1001	(AC0) ← INTEGER PART OF [(AC0), (AC1) ÷ (EA)]; (AC1) ← REMAINDER OF [(AC0), (AC1) ÷ (EA)]; OV; SEL ← 0; L altered	DIV disp(xr)

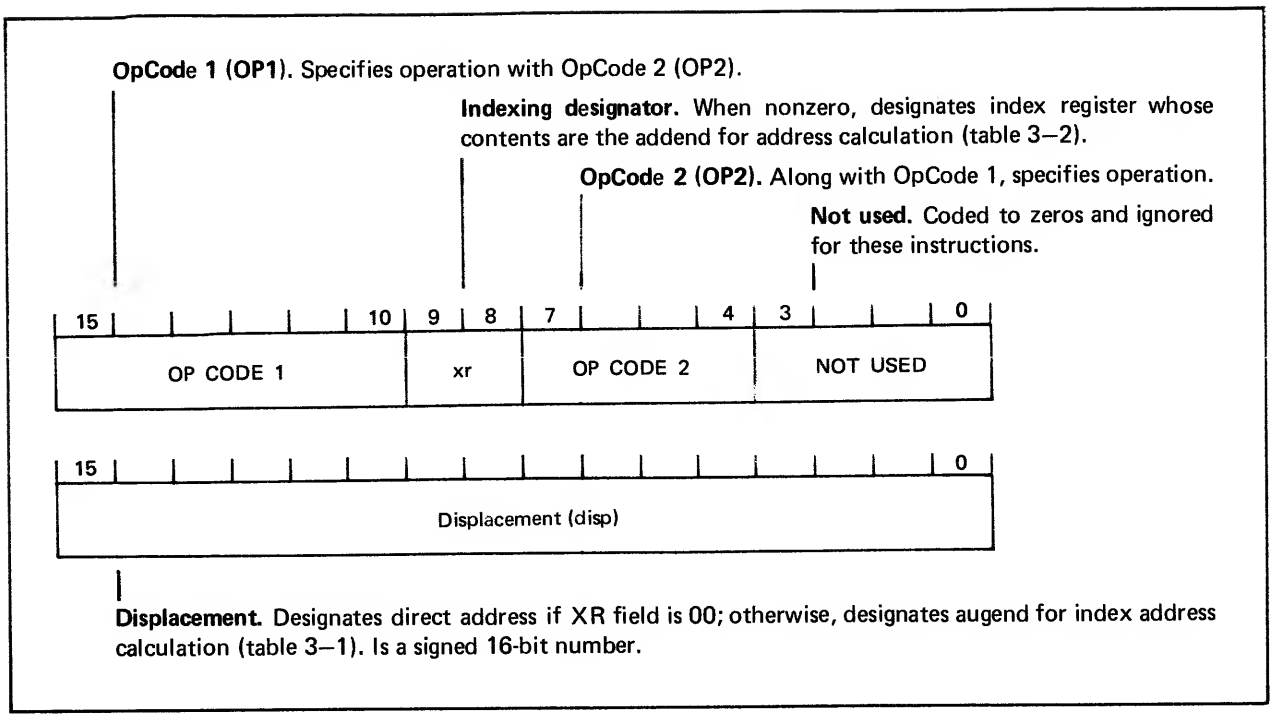
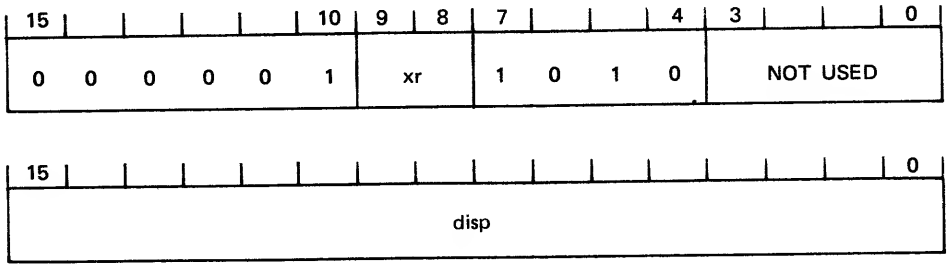


Figure 3–13. Double-Precision (Double-Word) Arithmetic Instruction Format

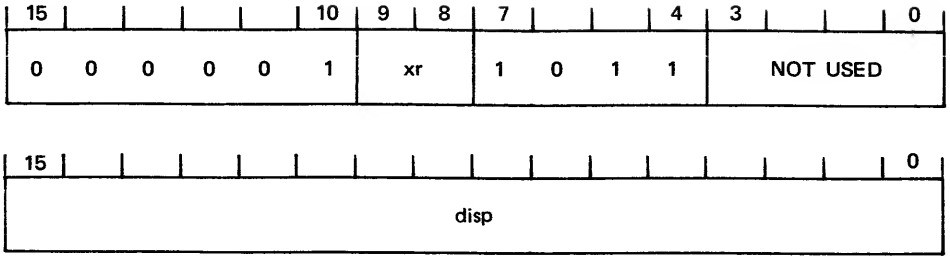
Double-Precision Add (DADD)



Operation: (AC0), (AC1) ← (AC0), (AC1) + (EA), (EA+1); OV; CY; SEL ← 0

Description: The double-precision twos-complement value in AC0 (high order) and AC1 (low order) is added to the double-precision twos-complement value in EA (high order) and EA+1 (low order), and the result is stored in AC0 and AC1. The contents of EA and EA+1 are unchanged. The overflow or carry flag is set if an overflow or carry occurs, respectively; otherwise, they are cleared. The select flag is cleared.

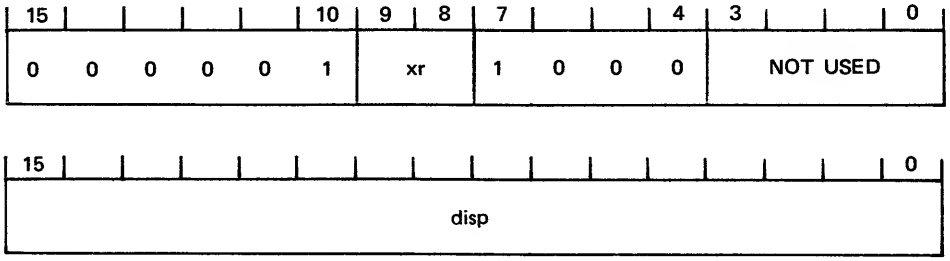
Double-Precision Subtract (DSUB)



Operation: (AC0), (AC1) ← (AC0), (AC1) + ∿ [(EA), (EA+1)] + 1; OV; CY; SEL ← 0

Description: The double-precision twos-complement value in EA (high order) and EA+1 (low order) is subtracted from the double-precision twos-complement value in AC0 (high order) and AC1 (low order). The contents of EA and EA+1 are unchanged. The overflow or carry flag is set if an overflow or carry occurs, respectively; otherwise, they are cleared. The select flag is cleared.

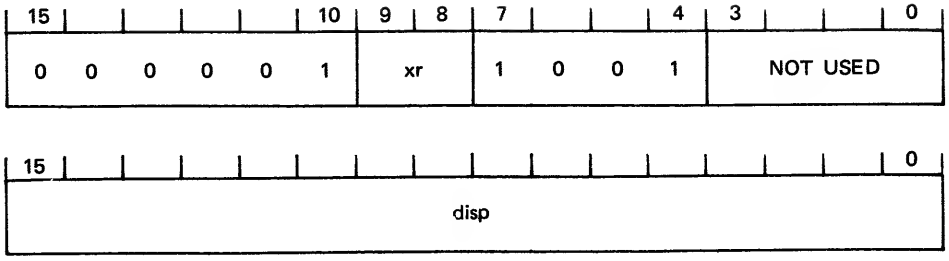
Multiply (MPY)



Operation: (AC0), (AC1) ← (AC1) * (EA); SEL ← 0; L altered

Description: The unsigned integer in AC1 is multiplied by the positive integer in the effective memory location EA. The high-order part of the 32-bit result is stored in AC0 and the low-order part is stored in AC1. The previous contents of AC0 and AC1 are lost. The contents of EA are unaffected. The select flag is cleared. The link flag is left in an arbitrary state.

Divide (DIV)



Operation: (AC0), (AC1) ← (AC0), (AC1) ÷ (EA); OV; SEL ← 0; L altered

Description: The positive 32-bit integer in AC0 (high-order part) and AC1 (low-order part) is divided by the contents (a positive number) of the effective memory location EA. The integer quotient is placed in AC1 and the remainder in AC0. The overflow flag (OV) is set if either of the following occurs: (1) the high-order part of the dividend (initial contents of AC0) is greater than or equal to the divisor, or (2) the quotient is negative. The select flag is cleared. The link flag is left in an arbitrary state. The contents of EA are unchanged. Division by zero, an illegal operation, falls into case 1 above.

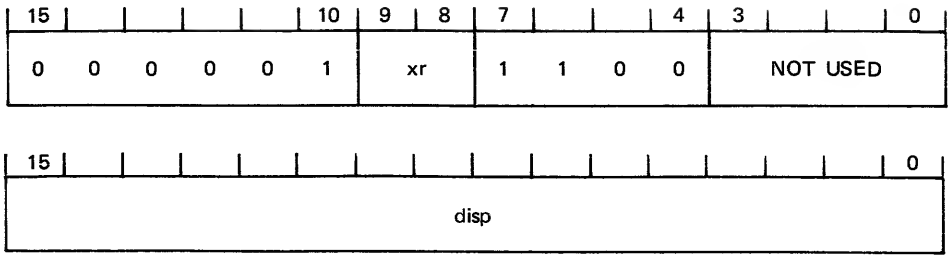
3.7.3 BYTE INSTRUCTIONS

There are two instructions in this category; both are double-word, memory reference instructions. They are summarized in table 3–15 and then individually described. Their word format is shown in figure 3–13.

Table 3–15. Byte Instructions

Instruction	Operation Code		Operation	Assembler Format
	OP1	OP2		
LOAD BYTE	000001	1100	(Low-order byte of AC0) ← byte from (EA ÷ 2); SEL ← 0	LDB disp(xr)
STORE BYTE	000001	1101	Byte of (EA ÷ 2) ← (low-order byte of AC0); SEL ← 0	STB disp(xr)

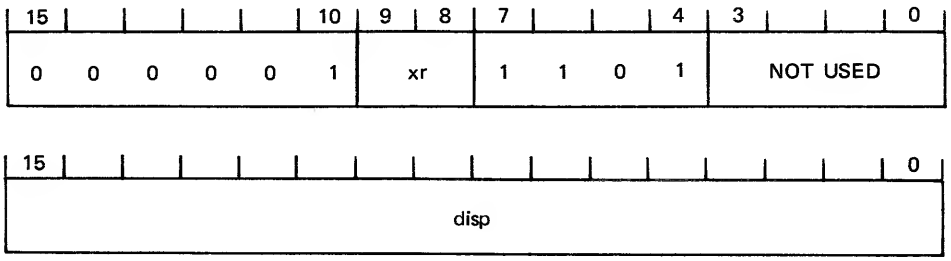
LOAD BYTE (LDB)



Operation: Low-order byte of (AC0) ← byte from (EA ÷ 2); SEL ← 0

Description: The low-order byte of AC0 is loaded with a byte from (EA ÷ 2). If the low-order bit of EA is 1, the low-order byte is loaded; otherwise, the high-order byte is loaded. (Note: EA ÷ 2 is the effective address shifted right one position.) The high-order byte of AC0 is set equal to zero. The select flag is cleared.

STORE BYTE (STB)



Operation: Byte of (EA ÷ 2) ← low-order byte from (AC0); SEL ← 0

Description: The low-order byte of AC0 is stored into the byte of (EA ÷ 2) specified by the low-order bit of EA. If the low-order bit is 1, the low-order byte is specified; otherwise, the high-order byte is specified. (Note: EA ÷ 2 is the effective address shifted right one position.) The unspecified byte of EA ÷ 2 and the contents of AC0 are unaffected. The select flag is cleared.

PROGRAMMING NOTE

The effective address is formed by adding the contents of the index register to the displacement (EA = XR + DISP). Byte addresses are formed by shifting this quantity right one bit position (EA ÷ 2 = [XR + DISP]/2 = XR/2 + DISP/2). Bit 0 of the EA specifies the left byte if equal to 1; the right bit if equal to zero.

3.7.4 BIT AND STATUS FLAG INSTRUCTIONS

There are seven single-word instructions in this group. They are summarized in table 3–16 and then described individually. Their word formats are shown in figure 3–14.

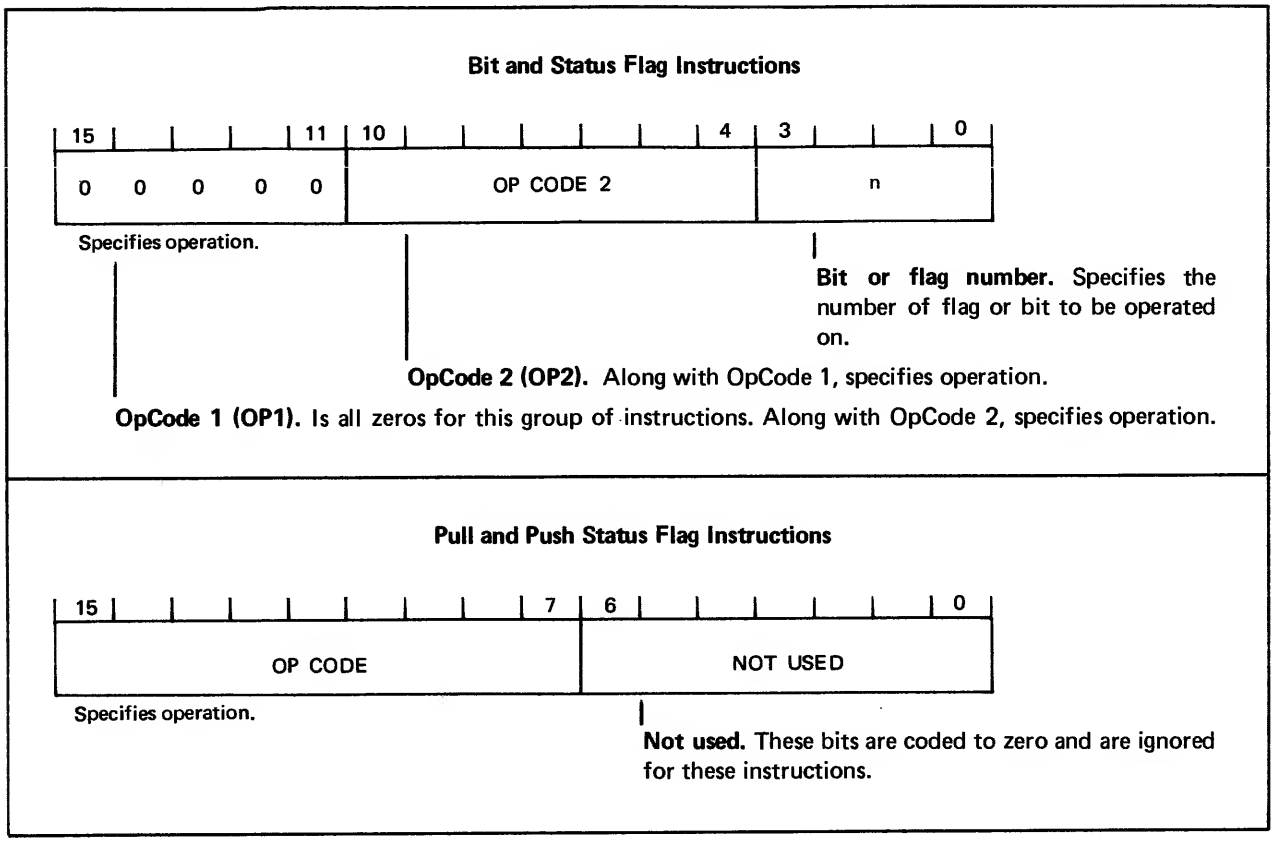
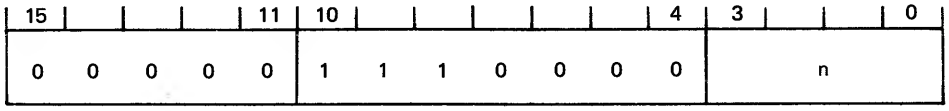


Figure 3–14. Bit and Status Flag Instruction Formats

Table 3–16. Bit and Status Flag Single-Word Instructions

Instruction	OpCode 2	Operation	Assembler Format
SET STATUS FLAG	1110000	Status Flag n ← 1; SEL ← 0	SETST n
CLEAR STATUS FLAG	1110001	Status Flag n ← 0; SEL ← 0	CLRST n
SET BIT	1110010	$AC0_n$ ← 1; SEL ← 0	SETBIT n
CLEAR BIT	1110011	$AC0_n$ ← 0; SEL ← 0	CLRBIT n
COMPLEMENT BIT	1110110	$AC0_n$ ← ~ $AC0_n$; SEL ← 0	CMPBIT n
SKIP IF STATUS FLAG TRUE	1110100	IF Status Flag n = 1, then (PC) ← (PC) + 1; SEL ← 0	SKSTF n
SKIP IF BIT TRUE	1110101	IF $AC0_n$ = 1, then (PC) ← (PC) + 1; SEL ← 0	SKBIT n

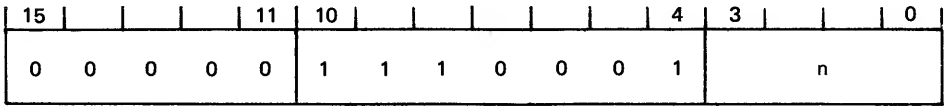
SET STATUS FLAG (SETST)



Operation: Status Flag n ← 1; SEL ← 0

Description: Bit n of the status flag register is set true. All other bits are unaffected. The select flag is cleared. ($0 \leqslant n \leqslant 15$)

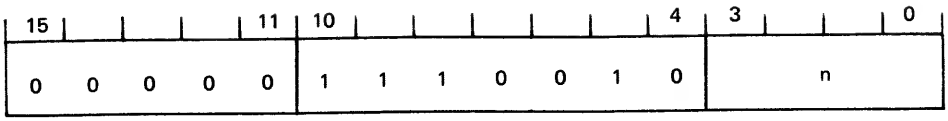
CLEAR STATUS FLAG (CLRST)



Operation: Status Flag n ← 0; SEL ← 0

Description: Bit n of the status flag register is cleared. All other bits are unaffected. The select flag is cleared. ($0 \leqslant n \leqslant 15$)

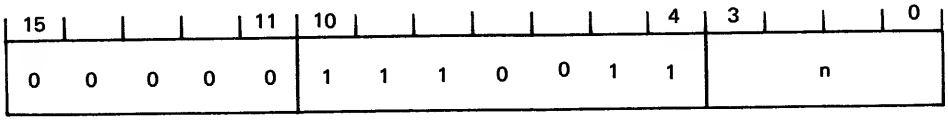
SET BIT (SETBIT)



Operation: $AC0_n \leftarrow 1$; SEL ← 0

Description: Bit n of AC0 is set true. All other bits are unaffected. The select flag is cleared. ($0 \leqslant n \leqslant 15$)

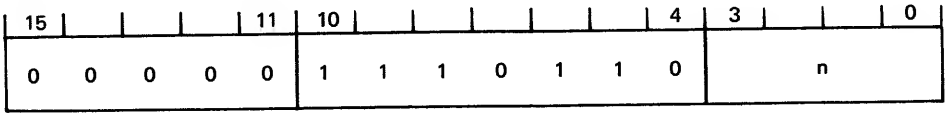
CLEAR BIT (CLRBIT)



Operation: $AC0_n \leftarrow 0$; SEL ← 0

Description: Bit n of AC0 is cleared. All other bits are unaffected. The select flag is cleared. ($0 \leqslant n \leqslant 15$)

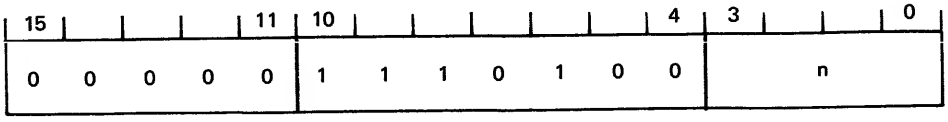
COMPLEMENT BIT (CMPBIT)



Operation: $AC0_n \leftarrow \sim AC0_n$; SEL ← 0

Description: Bit n of AC0 is complemented. All other bits are unaffected. The select flag is cleared. ($0 \leqslant n \leqslant 15$)

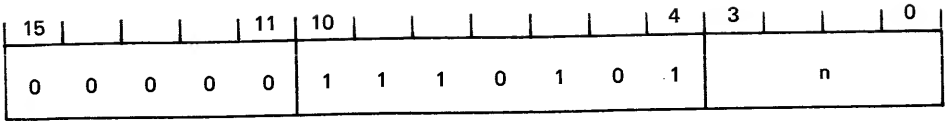
SKIP IF STATUS FLAG TRUE (SKSTF)



Operation: IF Status Flag n = 1, (PC) ← (PC) + 1; SEL ← 0

Description: If Status Flag n is true, the next memory location is skipped. The contents of the status flags are unaffected. The select flag is cleared.

SKIP IF BIT TRUE (SKBIT)



Operation: IF $AC0_n = 1$, (PC) ← (PC) + 1; SEL ← 0

Description: If bit n of AC0 is true, the next memory location in sequence is skipped. The contents of AC0 are unaffected. The select flag is cleared.

PROGRAMMING NOTE

Caution should be taken when coding a skip instruction to prevent the skip condition from jumping into the displacement field of a double-word instruction.

3.7.5 INTERRUPT HANDLING INSTRUCTIONS

There are two instructions in this group, listed in table 3–17. Their word format is shown in figure 3–15, and then described in succeeding paragraphs.

Table 3–17. Interrupt Handling Instructions

Instruction	OpCode 2	Operation	Assembler Format

INTERRUPT SCAN	1010001	If (AC1) = 0, SEL ← 0;
		If (AC1) ≠ 0, SEL ← 0;
		AC1 ← [shift AC1 right 1] until 1 shifted out
		(AC2) ← (AC2) + number of shifts; (PC) ← (PC) + 1
			ISCAN

JUMP INDIRECT TO LEVEL 0 INTERRUPT	1010010	(STK) ← (PC); (PC) ← ([120_{16} + disp]) INTEN ← 0	JINT disp

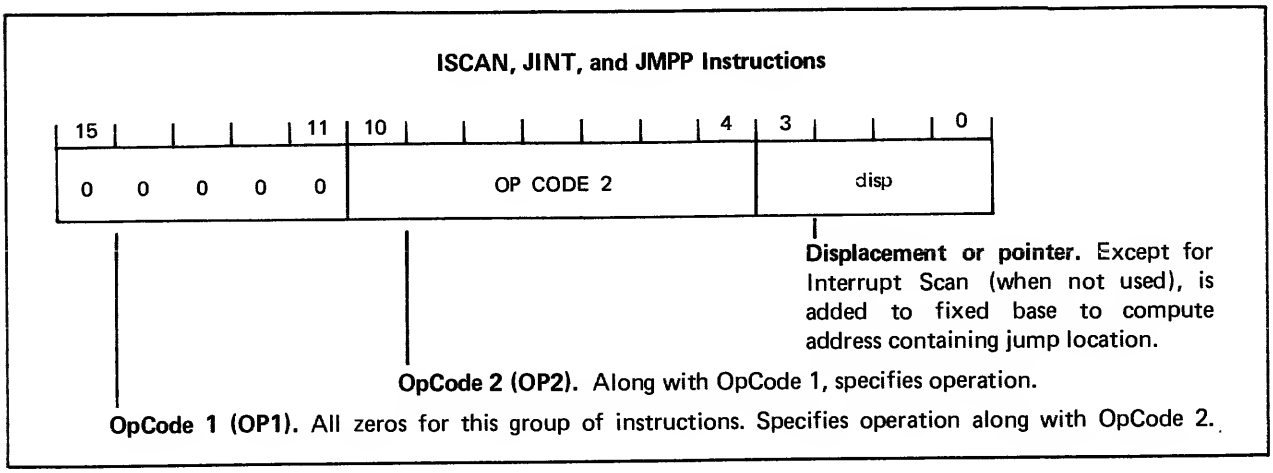
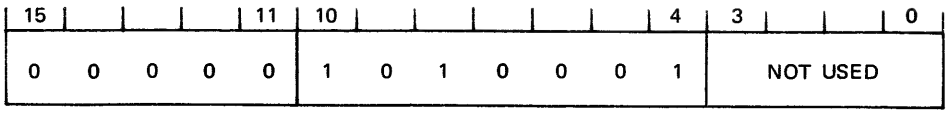


Figure 3–15. Interrupt and Word Jump Formats

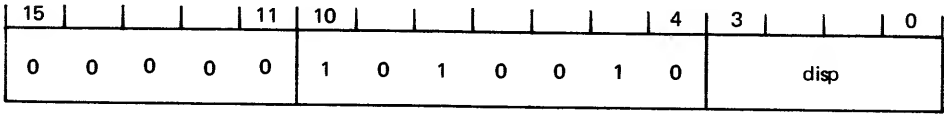
INTERRUPT SCAN (ISCAN)



Operation: If AC1 = 0, SEL ← 0;
Else, SEL ← 0;
(AC1) ← (shift AC1 right 1) until 1 shifted out;
(AC2) ← (AC2) + number of shifts;
(PC) ← (PC) + 1

Description: If AC1 = 0, the select flag is cleared and the next instruction is executed. If AC1 ≠ 0, then the select flag is cleared and AC1 is shifted right until a 1 is shifted out of bit 0. The number of shifts which occurred is added to the contents of AC2. The next memory location is skipped.

JUMP TO LEVEL 0 INTERRUPT, INDIRECT (JINT)



Operation: (STK) ← (PC); (PC) ← ([120_{16} + disp]); INTEN ← 0

Description: The contents of the PC are pushed onto the top of the stack. The new contents of the PC are set equal to the contents of the memory location whose address is formed by adding disp to 120_{16}. The interrupt enable flag is cleared.

3.7.6 TRANSFER-OF-CONTROL INSTRUCTIONS

There are two instructions in this group, summarized in table 3–18. The word formats are depicted in figures 3–15 and 3–16.

Table 3–18. Transfer-of-Control Instructions

Instruction	OP2	Operation	Assembler Format
JUMP THROUGH POINTER	1010000	(PC) ← (100_{16} + disp)	JMPP disp
JUMP TO SUBROUTINE THROUGH POINTER	0110	(STK) ← (PC); (PC) ← (disp + 100_{16})	JSRP disp

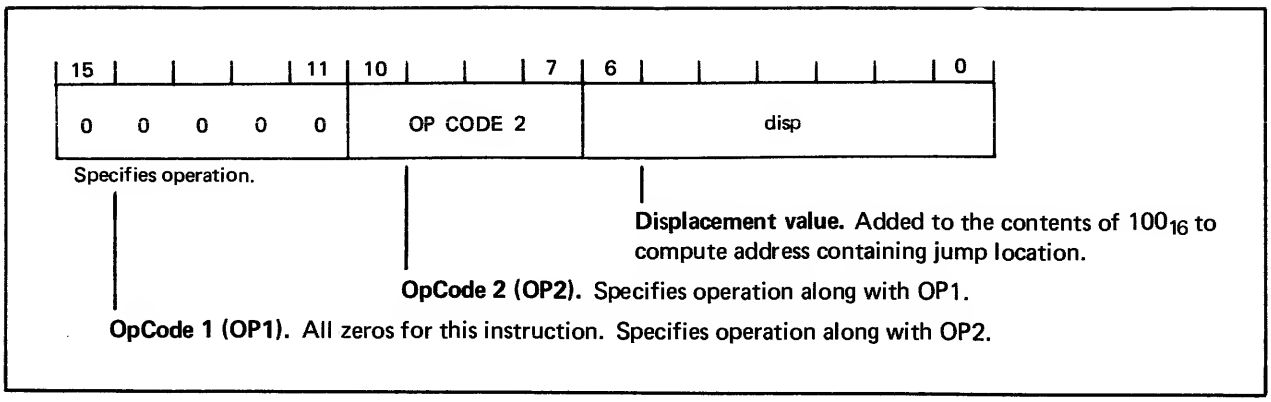
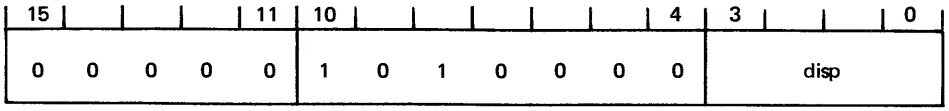


Figure 3–16. Jump to Subroutine Through Pointer Instruction Format

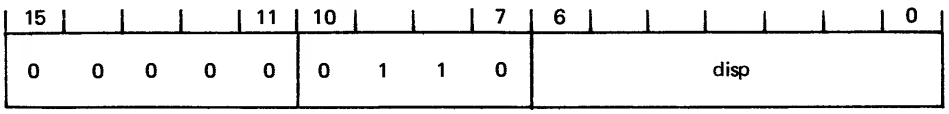
JUMP THROUGH POINTER (JMPP)



Operation: (PC) ← ([100_{16}) + disp])

Description: The contents of the PC are set equal to the contents of the memory location addressed by the sum of the contents of location 100_{16} and disp.

JUMP TO SUBROUTINE THROUGH POINTER (JSRP)



Operation: (STK) ← (PC), (PC) ← ([100_{16}) + disp])

Description: The program counter is pushed onto the top of the stack. The new contents of the PC are set equal to the contents of the memory location addressed by the sum of the contents of location 100_{16} and disp.

Chapter 4
CIRCUIT DESCRIPTIONS

This chapter treats each functional block of the IMP-16C at the circuit level, explaining the circuit operation and design configurations. Functional blocks are considered individually, and, at the end of this chapter, figure 4–6, an overall schematic diagram of the IMP-16C, is presented on three foldout sheets. These should be referred to during the circuit descriptions that follow. Each circuit described is shown in a broken-line enclosure having the name of the circuit.

Figure 4–7 is a functional block diagram of the IMP-16C circuits detailed on the schematic diagram of figure 4–6. Most of the units, the data flow, and the control functions are briefly explained in chapter 2 with reference to figure 2–1. The clock and timing circuits, refresh logic, system initialization, and jump/flag timing and control logic are further details added to figure 4–7 so it corresponds to and gives an exact overview of the actual circuits on the IMP-16C schematic diagram. The sheet number given in each functional block of figure 4–7 refers to the sheet number of figure 4–6 on which the circuit is detailed.

Figure 4–7 is on a foldout sheet located following figure 4–6 so it may be readily referenced from any part of this chapter or as a quick guide to the schematic diagram of figure 4–6.

A parts list and a component layout of the IMP-16C card are presented in table 4–2 and figure 4–8, respectively. (Table 4–2 and figure 4–8 are located on page 4–17/18.)

The descriptions that follow mention one CROM in the text; however, all discussions also apply to the case of two CROMs.

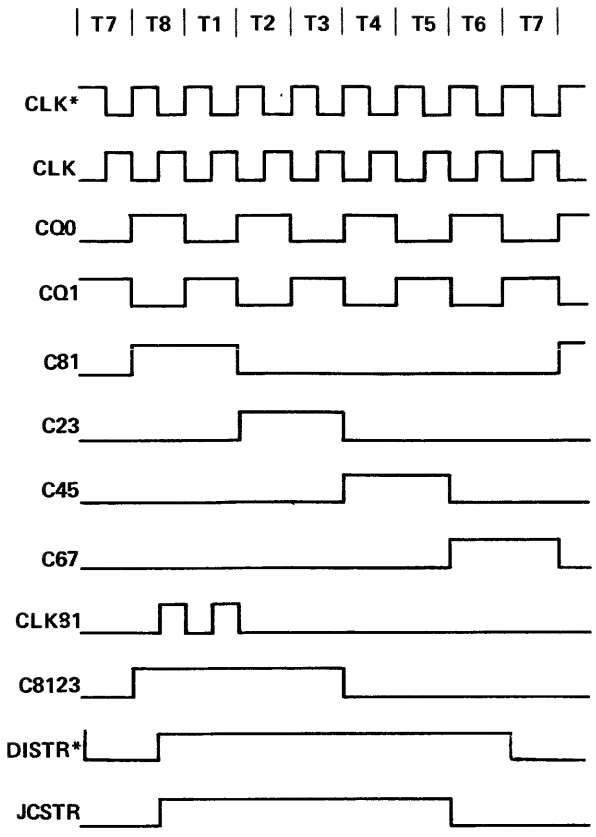
4.1 MASTER CLOCK AND 4-PHASE CLOCK GENERATORS *(Sheet 1, figure 4–6)*

The 4-phase clocks required for the CPU devices (one CROM and four RALUs) are generated with a shift register and two MH0026 clock drivers. The master clock signal is generated by a crystal oscillator circuit made from a DM10116 triple line receiver connected as an amplifier and a Schmitt trigger circuit. Two transistors, Q1 and Q2, provide level shifting to convert from ECL levels to TTL compatible logic signals.

The shift register DM74195 generates four clock signals, each of which lasts for two time periods. These signals are then logically gated to yield the odd phases that drive the MH0026 clock driver devices. The MH0026 clock drivers are capable of driving 1000 pf loads with rise and fall times of 20 ns. The typical loading by the CROM and the RALUs is 215 pf (45 pf for each device). The resistors in the output lines of the MOS clock drivers damp out any possible clock overshoots by compensating for the inductance of the clock lines.

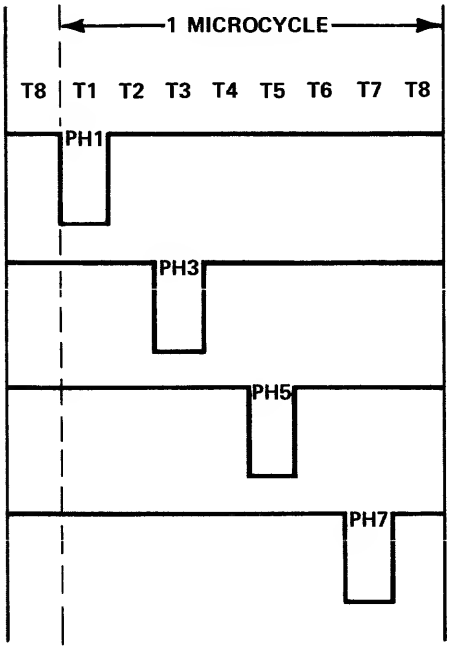
The DM74195 shift register outputs are also used to generate some of the other timing signals required in accordance with the timing diagrams shown in figure 4–1. These signals are derived by gating the appropriate shift register outputs with various combinations of the master clock and the shift register input clock.

In figure 4–1, the symbols used to designate clock periods have the following significance: the numbers following the letter C denote the specific time period for which the signal is valid. For example, C23 refers to a clock that is high during T2 and T3. Similarly, CLK81 refers to a signal derived from the logical AND of C81 and CLK. The MOS clocks and phase signals that drive the CPU circuits are shown in figure 4–2.



NS00129

Figure 4–1. IMP-16C Basic Timing Signals



NS00130

Figure 4–2. MOS Clocks and Phase Signals

4.2 MOS/LSI CPU LOGIC *(Sheet 3, figure 4–6)*

The CPU consists of one CROM and four RALU circuits driven by the 4-phase clocks. Control between the CROM and the RALUs is effected over the NCB (complemented control bus) lines. The DI (Data In) lines to the CROM serve the purpose of entering the instruction word bits 7 through 15 into the CROM. For sending out a 4-bit address to the Conditional Jump Multiplexer and the Control Flag latches, the lines JFA0 through JFA3 (bidirectional from the CROM) are used. The jump condition signal (NJCOND) enters the CROM at the same pin as bit 7 of the instruction word. The CROM has a flag enable signal[1] (NFLEN) that may be pulsed during T2 to set a particular control flag and/or may be pulsed at T6 to reset the flag (figure 2–4).

The other signals that go to and come from the CPU indicate various status conditions and also perform certain auxiliary operations. The following paragraphs explain these functions.

During an instruction fetch, bits 0 through 7 are loaded into the RALU Memory Data Register (figure 2–2), and bit 7 of the instruction word is extended through bit positions 8 through 15 of the Memory Data Register. In this way, the signed displacement value "disp" discussed in chapter 3 is extended for use in arithmetic operations for forming memory addresses and for immediate instructions. The SININ signal to the RALUs accomplishes the sign extension. For the two low-order RALUs (1 and 2), the SININ pins are permanently connected to a logic 0 (–12V). For the two high-order RALUs (3 and 4), where bits 8 through 15 are located, the SININ pins are connected to the bit 7 output of the Buffered Data Out lines; this bit 7 is used in the two high-order RALUs (3 and 4) to effect the sign extension of "disp" of the instruction word.

The STF signal indicates a "stack-full" condition. When the bottom entry of the stack is filled with nonzero data, the STF line is a "1." The STF lines of all RALUs are tied together and connected to the Conditional Jump Multiplexer to allow testing for the stack-full condition. A similar scheme is used to detect a zero-result condition with the NREQ0 signal. The NREQ0 lines are tied together for all the RALUs; the NREQ0 signal is a "0" if the R-bus is zero as a result of the preceding machine cycle.

During T7 and T8, CSH3 and CSH0 are used to transfer shift data: for a left shift, the most significant bit is shifted out over CSH3, and the least significant bit is shifted in by CSH0; for a right shift, the converse is true.

Each RALU has four status flags, which are interfaced to the A- and R-buses. This provides a convenient means of saving status after an interrupt and for setting the status flags. For all except the most significant RALU (4), the status flags are general purpose and may be used for a variety of functions, depending on the application requirements. For the most significant RALU (4), the status flags have the following functions:

LINK Flip-Flop. When the SEL input to the RALU is "1," the Link Flag (L) is included in shift operations.

OVERFLOW Flag. When enabled (under control of the CROM), the Overflow Flag (OV) is set if an arithmetic overflow occurs during an add operation.

CARRY Flag. When enabled (under control of the CROM), the Carry Flag (CY) is set to the value of the carry bit out of the most significant ALU bit after an add operation (figure 2–2).

FLAG Flip-Flop. This flag is available for general-purpose use.

These status flags may be loaded from the R-bus or stored onto the A-bus under control of the Save/Restore Flag (SVRST) input; this is used by the CROM to implement the PUSHF and PULLF instructions. The output of the general-purpose Flag is available at the Flag output pin; Carry and Overflow Flags are available at CYOV. The Select Flag (SEL) input is used to select the Carry or Overflow for output on CYOV and to determine whether the Link (L) is included in shift operations (discussed in chapter 3). General-purpose flags 0 and 12 are brought out to terminals on the IMP-16C edge connector as signals FLAG0 and FLAG12. The next section describes the flag logic circuits.

1 - The prefix "N" to a signal name denotes logical complementation in the MOS/LSI CROM and RALUs. For signals generated external to these units, an asterisk (*) suffixed to the signal name denotes complementation.

4.3 CONTROL FLAGS AND CONDITIONAL JUMP MULTIPLEXER LOGIC *(Sheet 2, figure 4–6)*

External to the CPU portion of the IMP-16C is the logic required to set and reset the control flags and to select one of 16 jump conditions.

The flag addresses sent out by the CROM are latched to keep them stable; this is done with a DM9322 multiplexer connected as a latch by feeding the outputs back to the second set of inputs. (This particular technique has been chosen here because the DM9322 has less propagation delay than conventional DM7475 or DM74175 latches.) The CROM sends out the flag addresses at T1; these are latched in the DM9322 device during the latter half of T1 by the signal CLK81.

The latched addresses are then used to select one of 16 jump conditions in a DM8219 16-to-1 multiplexer. The complemented flag enable signal (NFLEN), which is low at T2 and then again at T6, enables the selection of a flag in one of the two DM9334 8-bit addressable latch devices. The data to the addressable latches comes from the signal C8123, which provides a logic "1" when NFLEN is low at T2 and a logic "0" when NFLEN is low again at T6. This allows setting and resetting of the flags (figure 2–4) under control of NFLEN at T2 and T6, respectively.

The output of the TRI-STATE DM8219 device is tied directly to the jump condition (NJCOND) input of the CROM. This is the line that is tested during conditional jump operations; the testing is done during T2 (figure 2–4). The START and JC12 through JC15 inputs are user-supplied signals and could be asynchronously generated. Thus, in order to ensure that the logic levels for these signals are stable during T2, synchronizing latches are provided.

The various conditions that can be tested are hardwired to the conditional jump multiplexer according to table 3–8 in chapter 3. Four user-assigned jump conditions and six user-assigned control flag lines are brought out to pins on the edge connector.

The more significant 8 of the 16 control flags may be set using the SFLG instruction and cleared or pulsed using the PFLG instruction; the assignments of these flags (F8, INT EN, SEL, and F11 through F15) are listed with their flag codes (FCs) in table 3–13. The SEL control flag affects shift operations as described in 3.6.6, and also selects CY or OV for output on the CYOV line (see table 3–8). The less significant 8 flags are affected by the CROM-resident microprograms. The assignment of these flags is listed in table 4–1.

Table 4–1. Control Flags Affected by Microprogram

Flag Number	Signal Name	Function
0	RDM	Read Memory
1	WRM	Write Memory
2	RDP	Read Peripheral
3	WRP	Write Peripheral
4	CPINP	Control Panel Input Flag
5	SVRST	Save/Restore Status Flags
6	LDAR	Load Address Register
7	HLT	Set by HALT Instruction

4.4 INPUT MULTIPLEXER, DATA BUFFER, AND ADDRESS LATCHES *(Sheet 3, figure 4–6)*

The 16-bit bidirectional data bus from the RALU devices is used to transfer all information between the CPU and memory and peripheral units. This bus is buffered by passing all output signals through a set of TRI-STATE DM8095 hex-buffer circuits.

Input data destined for the CPU are passed through a set of input multiplexers such that data from a memory or peripheral unit may be switched in. The TRI-STATE DM8123 multiplexers are controlled by RDM, the read memory flag (delayed until T7 because data may be accepted into the CPU only at T7, as shown in figure 2–4).

During T7, the data lines are used for input to the RALU from system memory or peripheral devices. The data receivers are "zeros catching," so the data lines must not be allowed to go negative during T7 unless the data input is to be a zero. Because of this zeros catching feature, the strobe signal for input data (DISTR*) is generated such that it occurs during the latter half of T7; this ensures that the data bits will be strobed in only when it is assured that they are stable.

When the data bus is sending out an address during read and write operations, the address is stored in a register consisting of four DM8551 quad-D flip-flops. These address lines are brought out to terminals on the edge connector to be used for addressing peripheral devices and add-on memory. Data from the RALU are sent out during part of T3 and all of T4 (figure 2–4) and are clocked into the latch if the RDM and the LDAR flag is set. The RDM and LDAR flags are pulsed (set at T2 and reset at T6) during memory read and write operations. The outputs of the DM8551 devices may be disabled by logically controlling the ODIS line. If ODIS is taken to a logic "1," the bus is disabled.

4.5 READ/WRITE AND READ-ONLY MEMORIES *(Sheet 4, figure 4–6)*

When executing a memory read operation, the processor sends out an address on the RALU data input/output bus; this address starts coming up during T3, and it is assured of being valid during T4. In the IMP-16C, the address is strobed into a latch at T4. If the processor is used with slow memories whose access times are longer than the interval between T4 and T7, it is necessary to stretch a clock period to allow for the slow access. For this purpose, the circuit clock phase-4 stretcher (sheet 1, figure 4–6) is used to extend T4 for an additional two periods.

During read and write operations, a clock hold signal (HCLK) is developed during C3. The timing relations are given in figure 4–3. This signal sets a flip-flop output (HOLD) such that a count-by-four circuit is enabled. After four counts of the master clock, the HOLD flip-flop is shut off. The delay provided by the counter circuit is used to inhibit phase 4 of the clock generator circuit.

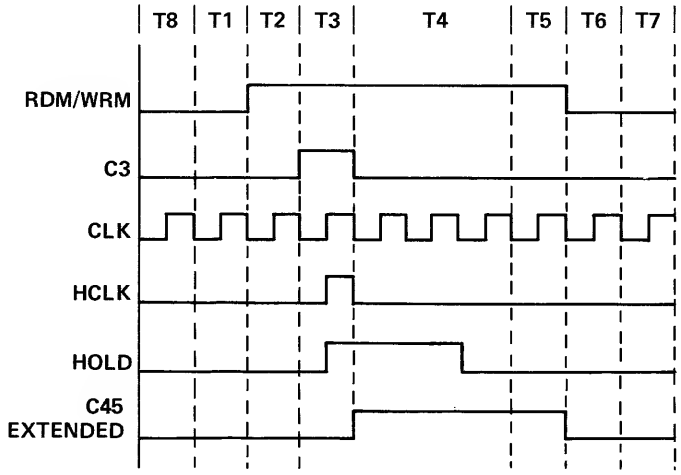


Figure 4–3. Timing Relations for Clock Hold Function

On-board address decoding for the memory is arranged such that address bit 15 controls the selection of the read-write and read-only memories. Bits 9 through 14 are ignored. Thus, if bit 15 is a logic "1" the ROMs are selected, and if bit 15 is a logic "0" the RWM is selected. If it is desired to change this arrangement or to use add-on memory, external address decoding must be done and appropriate signals supplied to the CS0, CS1, and CS2 pins of the IMP-16C. This option is explained in chapter 9.

4.6 INTERRUPT HANDLER *(Sheet 1, figure 4–6)*

The IMP-16C interrupt facility is handled through the conditional jump multiplexer inputs. Two interrupt inputs are provided. One is directly wired to the Conditional Jump Multiplexer and responds to a specific interrupt by jumping to a microprogram subroutine designed for control panel interrupts, as described in chapter 6. The other interrupt is a general interrupt input (INTRA), which can be wired to the user's interrupting device. The processing of interrupts is described in chapter 6.

The flip-flop output (INT-Q1) is set high whenever an external interrupt (INTRA) or a stack-full signal (STFL) is true simultaneously with the interrupt enable (INTEN) flag. INT-Q1 is wired to the interrupt input of the conditional jump multiplexer. The interrupt processing microprogram resets the interrupt enable (INTEN) flag to zero to disable any further interrupts, and control is transferred to the instruction stored in location 1 of main memory. The stack-full line is also wired to the jump condition multiplexer to permit testing for stack-full interrupts. If STFL causes such an interrupt, the bottom entry of the hardware stack is lost. In anticipation of this, the user can put a dummy word in the stack during his program initialization sequences.

See 6.3 for an explanation of the CPINT (Control Panel Interrupt) function.

4.7 SYSTEM INITIALIZATION *(Sheet 1, figure 4–6)*

During startup, the IMP-16C is initialized so all the sequential logic is conditioned to known logic states. There are two aspects to initialization: startup of the TTL logic and initialization of the CPU MOS/LSI devices.

The System Initialization circuit shows the startup logic required for the TTL clocks. When power comes on, the output of flip-flop (INIT*) is forced to a logic "0" (independent of the state of the SYSCLR* input) by the RC timing circuit. Because this signal goes to the Clear inputs of all the other flip-flops in the clock-generator circuit, the system starts up in a cleared condition. The system may also be cleared at any other time by grounding the SYSCLR* input.

When the System Clear signal (SYSCLR*) goes high, INIT* comes up after a delay of a few hundred milliseconds (time constant R10C5). At this time, all the system clocks are enabled. For systems that do not have external initialization, the SYSCLR* input should be left continuously at a logic "1."

Startup for the MOS parts is achieved by controlling the application of the –12-volt supply. The CPU MOS/LSI devices receive –12 volts from the SVGG (switched VGG) line. This voltage must be turned on a few milliseconds before the clocks are started. During turn-off, the clocks are kept on for a few milliseconds after SVGG goes off. The timing relationship between SVGG and the Power-On Condition (POC) signal in the System Initialization circuit is shown in figure 4–4.

Figure 4–5 is a recommended circuit that may be used to effect system reinitialization to clear the CPU MOS/LSI devices without shutting down other circuits on the IMP-16 card. This circuit is not on earlier versions of the IMP-16C card and would have to be user-supplied and connected to the SVGG terminal pin (sheet 2, figure 4–6). All IMP-16C cards with the part number 5511962 have the initialization circuit on board.

If a special reinitialization circuit, such as discussed above, is not supplied, the SVGG pins on the IMP-16C card-edge connector should be connected to the –12-volt supply. With this latter setup, reinitialization is effected by turning off power to the IMP-16C for at least 5 seconds and then turning it on again.

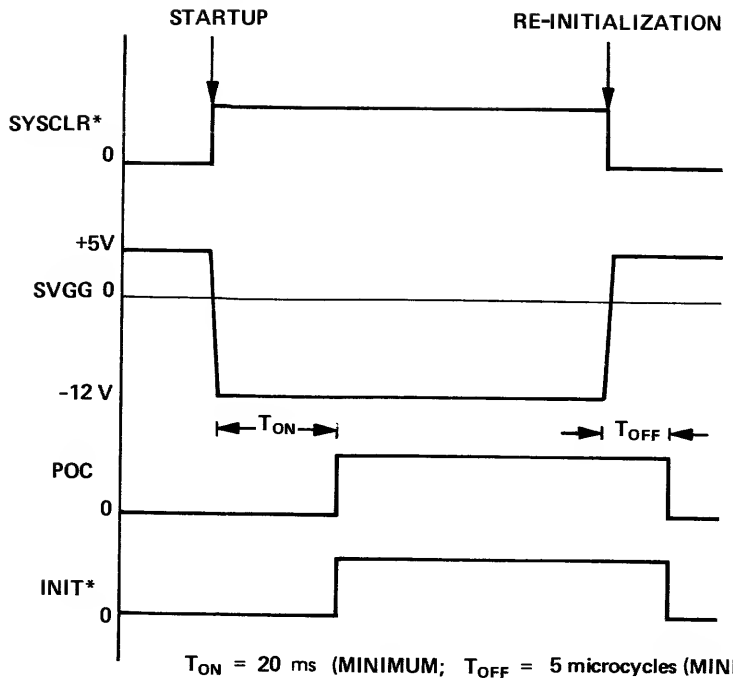
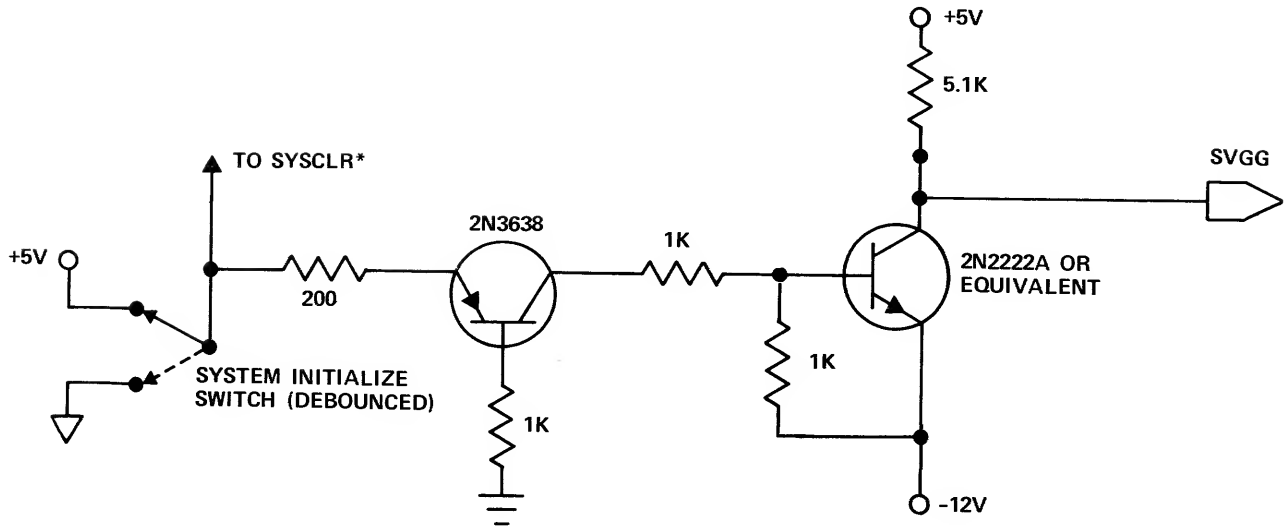


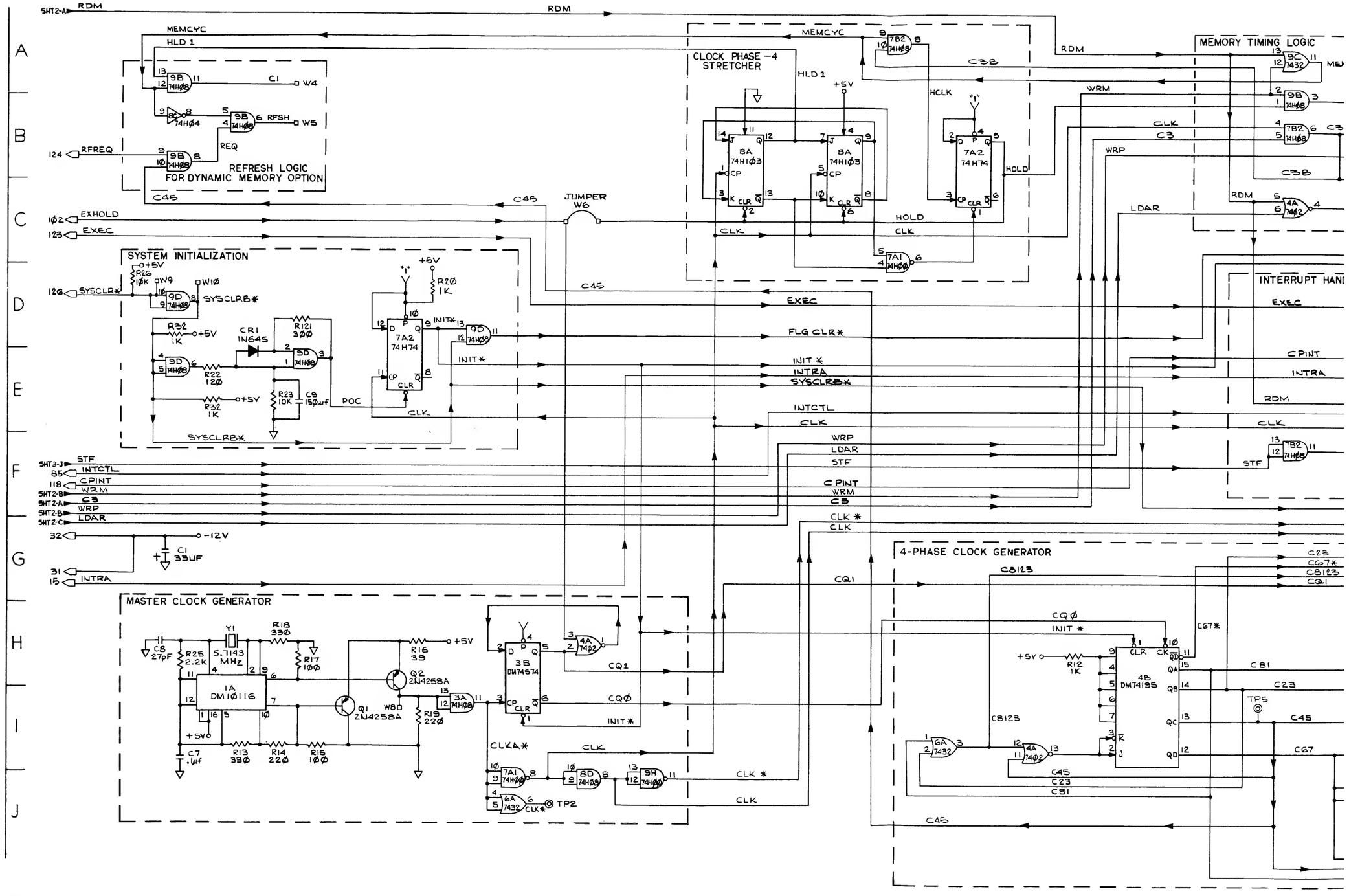
Figure 4–4. System Startup Timing



NOTE: BY TYING THE SYSCLR* PIN ON THE IMP-16C CARD TO A "SYSTEM INITIALIZE" BUTTON, THE SAME SIGNAL CAN BE USED TO CONTROL THE SWITCHED –12V (SVGG) FOR THE MOS/LSI DEVICES ON THE IMP-16C CARD. IN THIS CASE, IT IS REQUIRED THAT SYSCLR* BE LOW WHEN POWER IS APPLIED.

NS00133

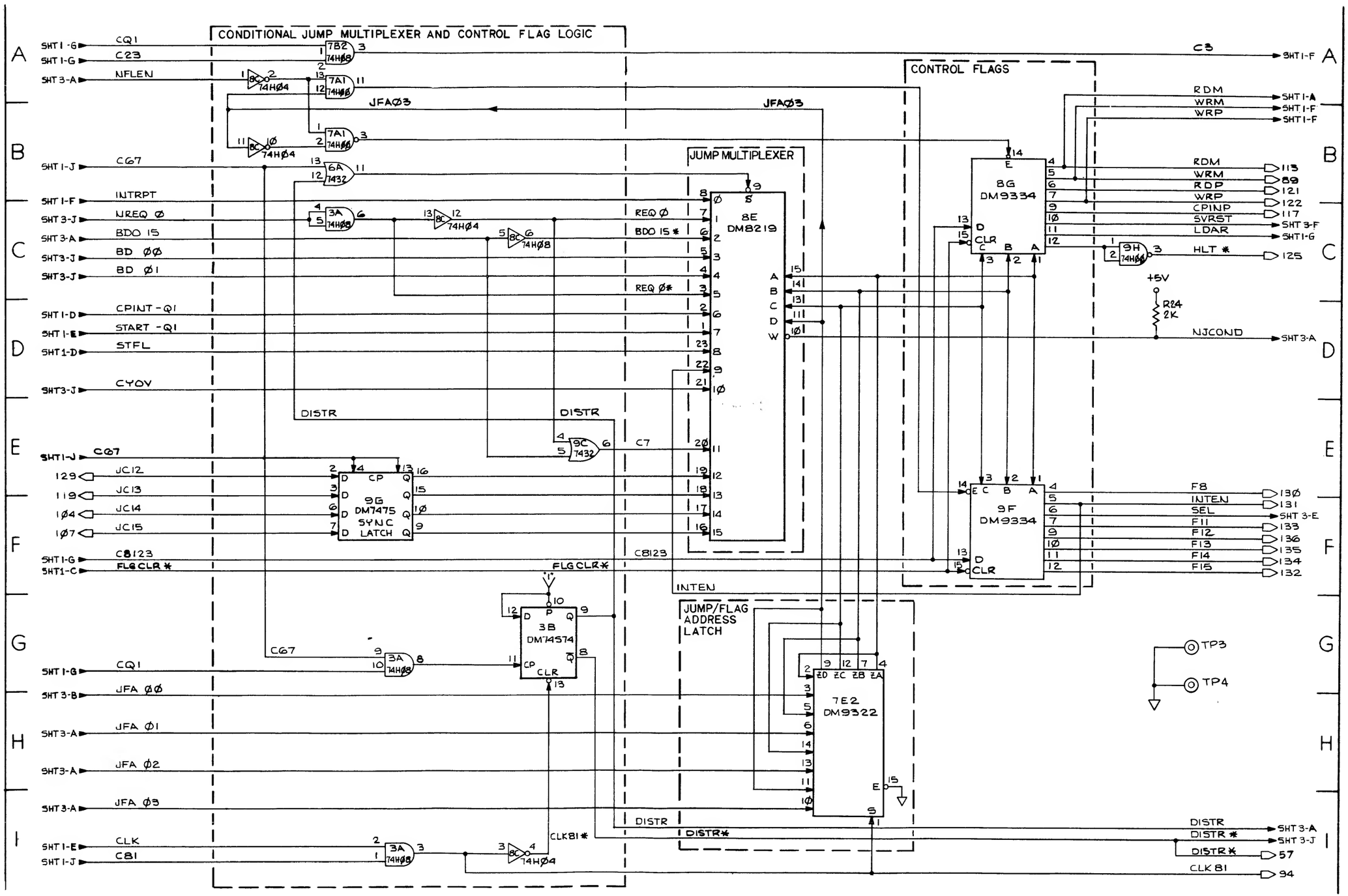
Figure 4–5. Circuit for Powering Up CPU MOS/LSI Devices



NOTE: INFORMATION ON THIS DIAGRAM IS SUBJECT TO CHANGE WITHOUT NOTICE.
ENGINEERING DIAGRAMS SUPPLIED WITH IMP-16C SHOULD BE REFERENCED.

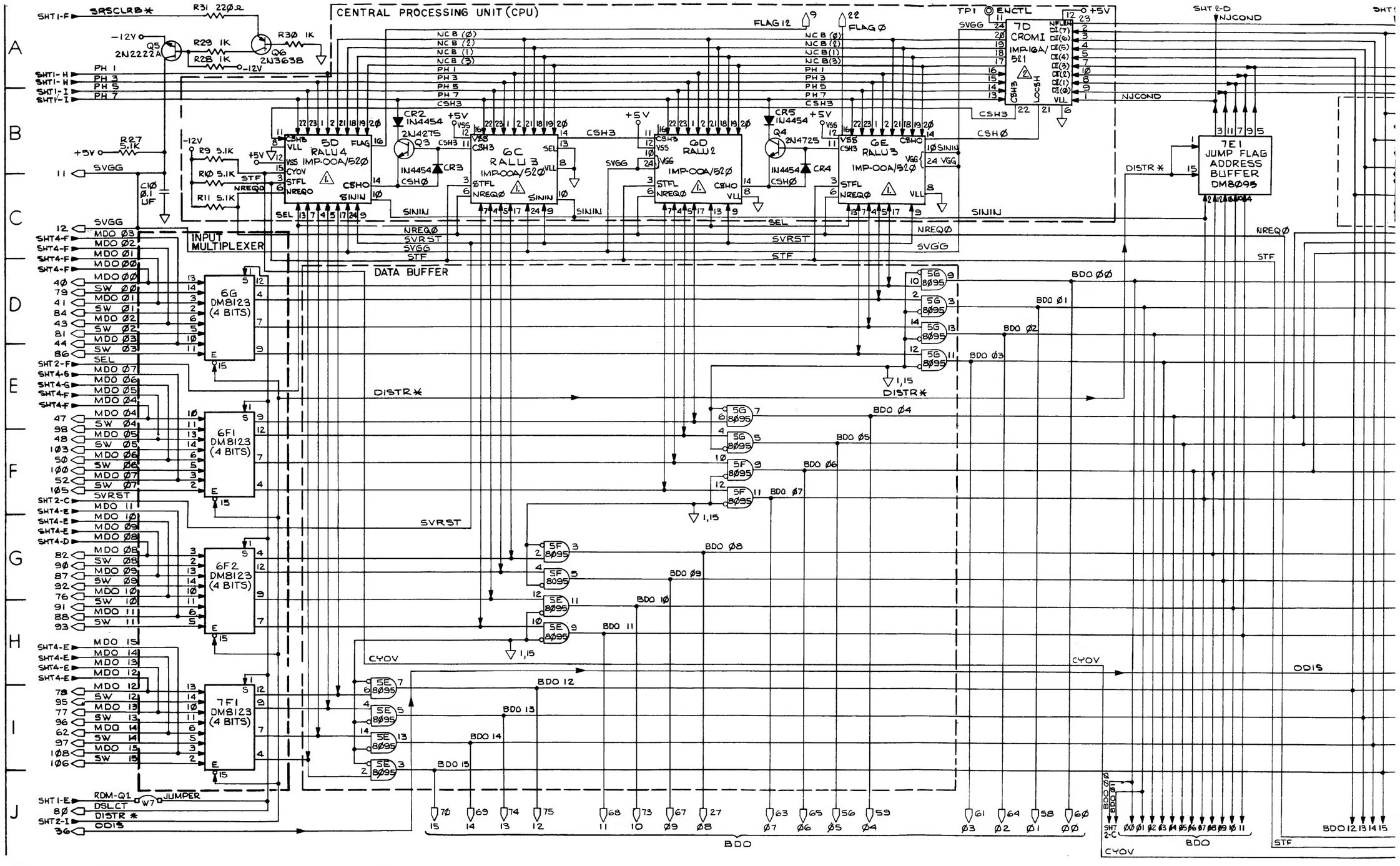
ON ON THIS DIAGRAM IS SUBJECT TO CHANGE WITHOUT NOTICE.
G DIAGRAMS SUPPLIED WITH IMP-16C SHOULD BE REFERENCED.

Figure 4–6. IMP-16C Schematic Diagram (sheet 1 of 4)

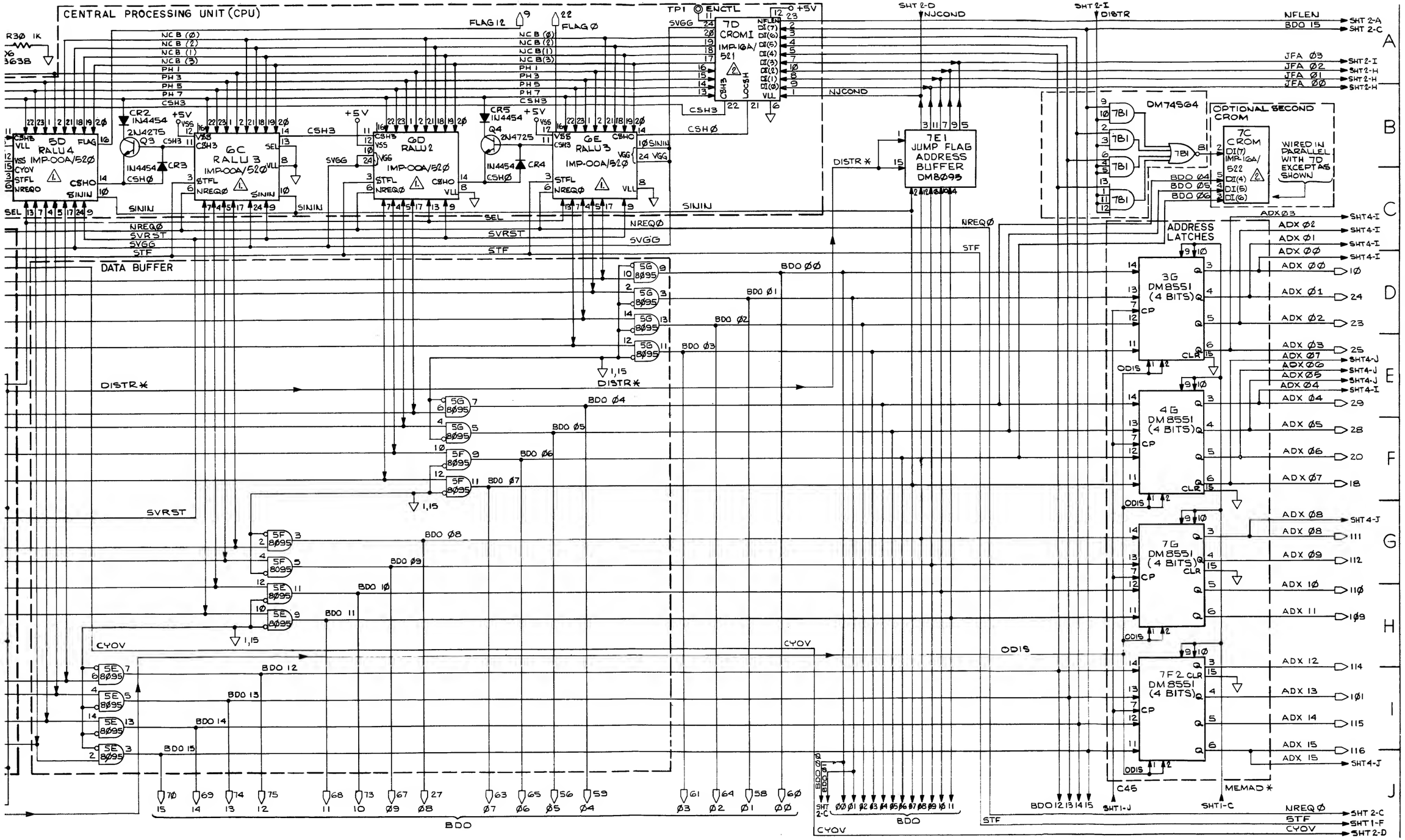


NOTE: INFORMATION ON THIS DIAGRAM IS SUBJECT TO CHANGE WITHOUT NOTICE. ENGINEERING DIAGRAMS SUPPLIED WITH IMP-16C SHOULD BE REFERENCED.

Figure 4–6. IMP-16C Schematic Diagram (sheet 2 of 4)



NOTE: INFORMATION ON THIS DIAGRAM IS SUBJECT TO CHANGE WITHOUT NOTICE.
ENGINEERING DIAGRAMS SUPPLIED WITH IMP-16C SHOULD BE REFERENCED.



1	NJCND	VGG(-12V)	24
2	DI(7)	NFLEN	23
3	DI(6)	HOCSH	22
4	DI(5)	LOCSH	21
5	DI(4)	NCB(Ø)	20
6	(GND)VLL	NCB(2)	19
7	DI(3)	NCB(1)	18
8	DI(1)	NCB(3)	17
9	DI(Ø)	Ø1	16
10	DI(2)	Ø3	15
11	ENCTL	Ø5	14
12	(+5)VSS	Ø7	13

CROM

IMP-16A/521

2 TYP CROM I OR II

1	Ø3	VGG(-12V)	24
2	Ø1	Ø5	23
3	STFL	Ø7	22
4	DATA(2)	NCB(Ø)	21
5	DATA(1)	NCB(3)	20
6	NREQØ	NCB(1)	19
7	DATA(3)	NCB(2)	18
8	(GND)VLL	DATA(Ø)	17
9	SVRST	FLAG	16
10	SININ	CYOV	15
11	CSH3	CSHO	14
12	(+5)VSS	SELECT	13

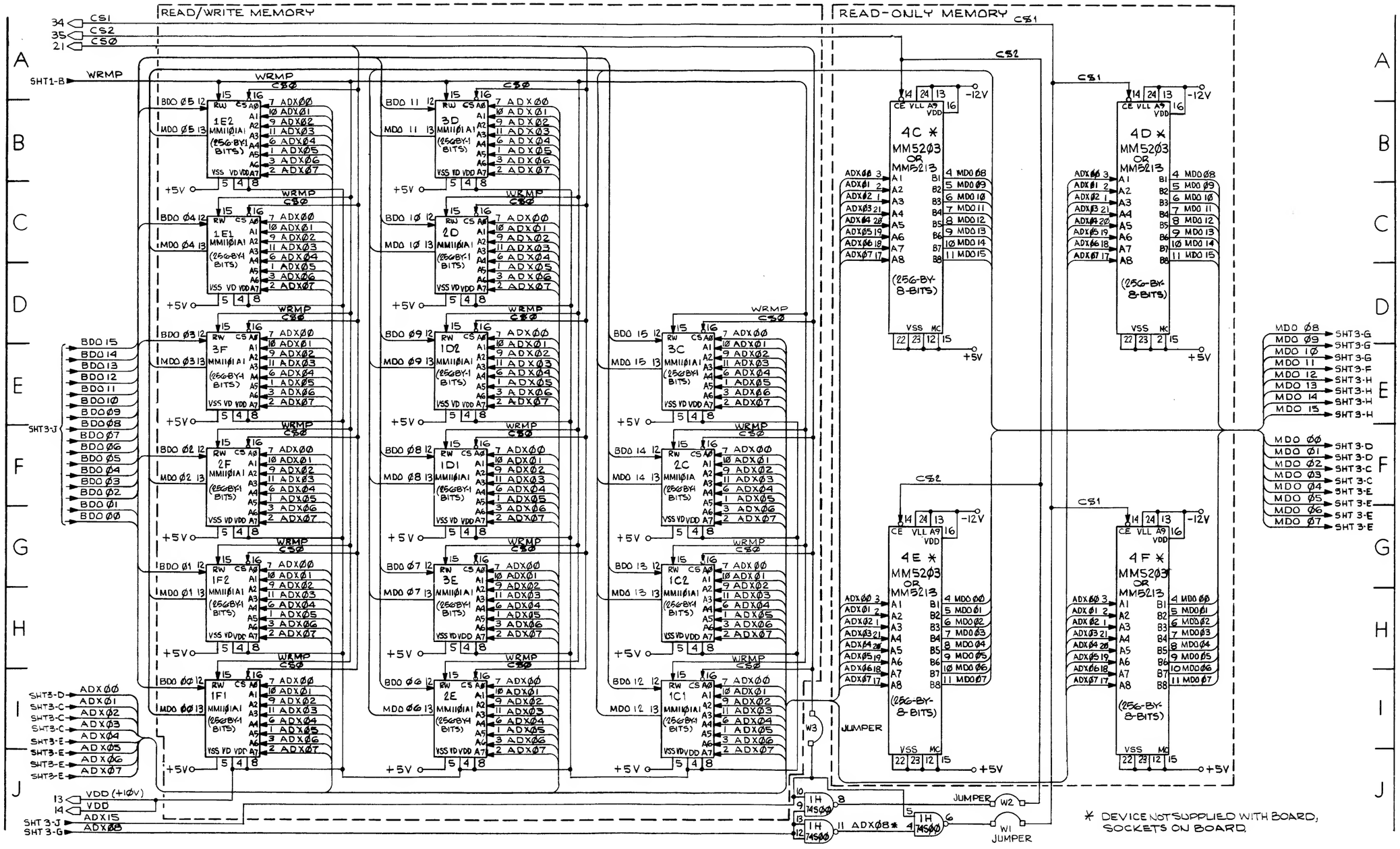
RALU 1-4

IMP-00A/520

1 TYP 5D,6C,6D&6E

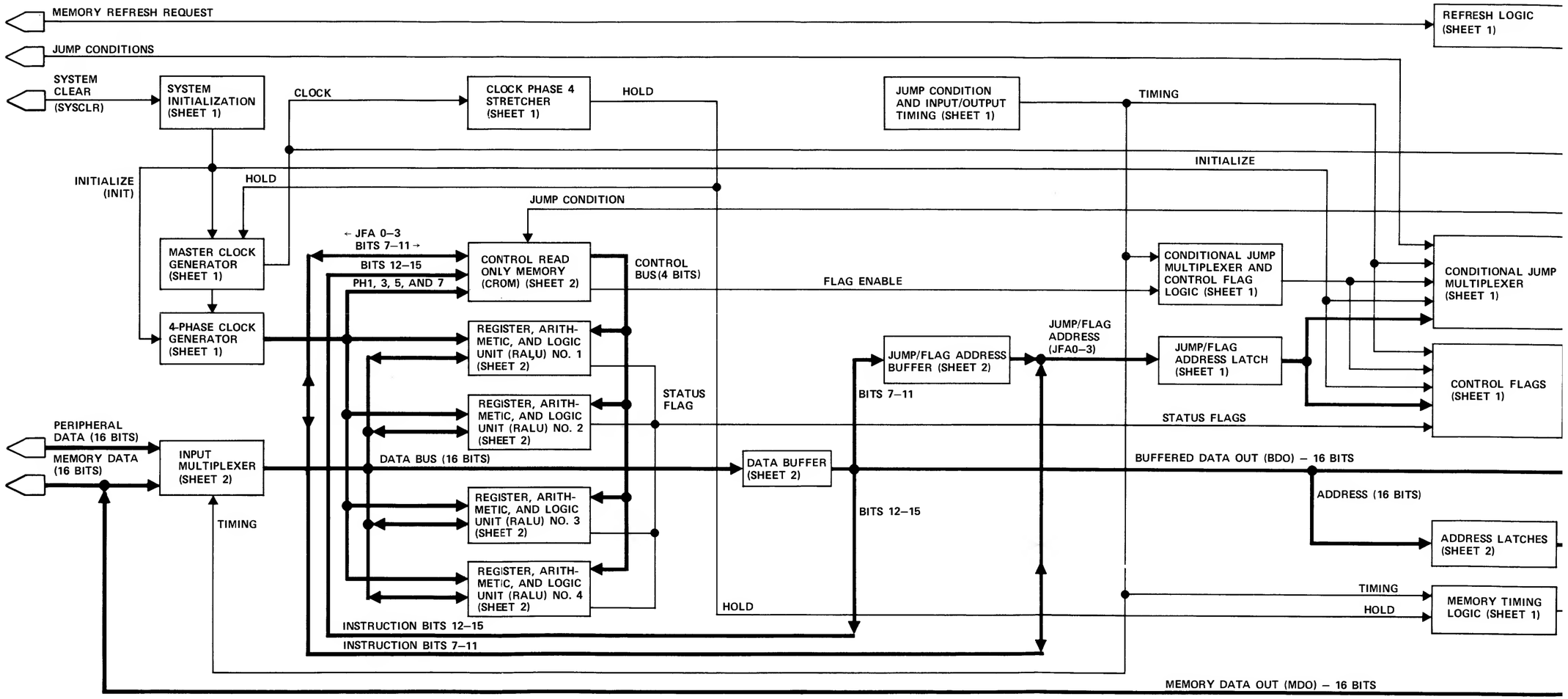
IS SUBJECT TO CHANGE WITHOUT NOTICE.
D WITH IMP-16C SHOULD BE REFERENCED.

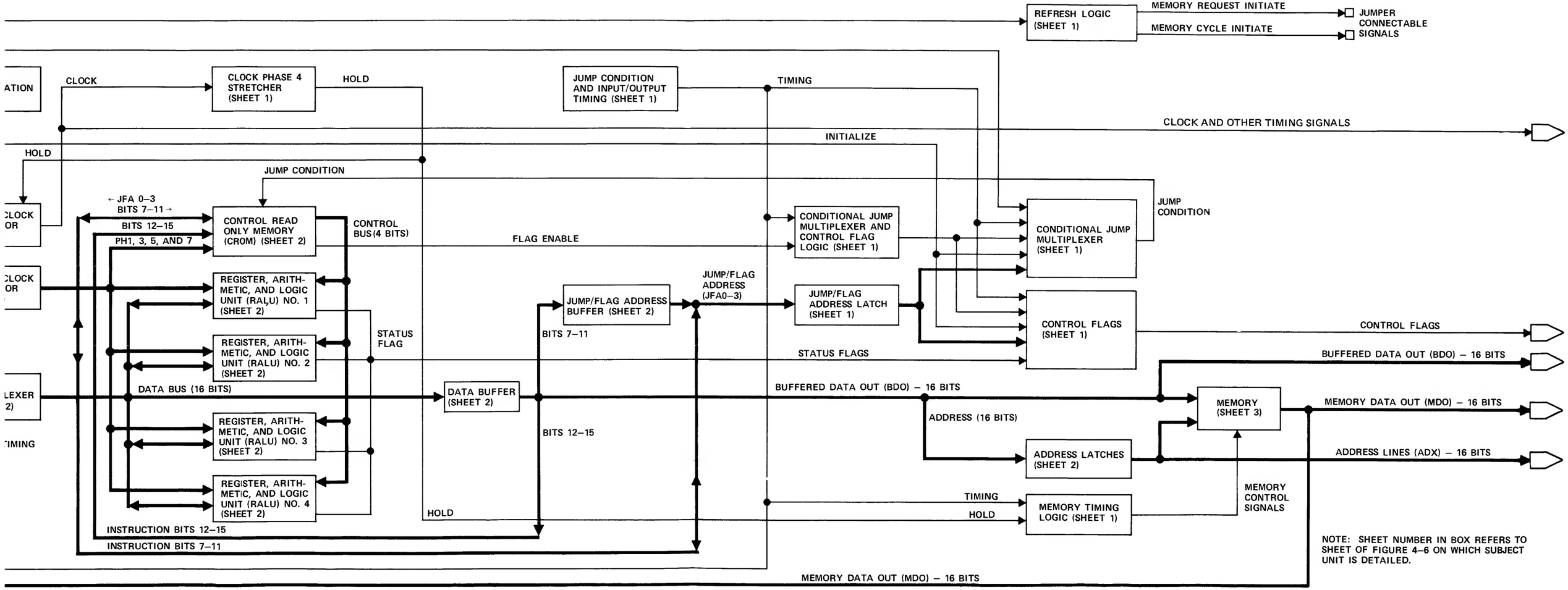
Figure 4–6. IMP-16C Schematic Diagram (sheet 3 of 4)



NOTE: INFORMATION ON THIS DIAGRAM IS SUBJECT TO CHANGE WITHOUT NOTICE. ENGINEERING DIAGRAMS SUPPLIED WITH IMP-16C SHOULD BE REFERENCED.

Figure 4–6. IMP-16C Schematic Diagram (sheet 4 of 4)


MEMORY REFRESH REQUEST
REFRESH LOGIC (SHEET 1)
JUMP CONDITIONS
SYSTEM CLEAR (SYSCLR)
SYSTEM INITIALIZATION (SHEET 1)
CLOCK
CLOCK PHASE 4 STRETCHER (SHEET 1)
HOLD
JUMP CONDITION AND INPUT/OUTPUT TIMING (SHEET 1)
TIMING
INITIALIZE
INITIALIZE (INIT)
HOLD
JUMP CONDITION
MASTER CLOCK GENERATOR (SHEET 1)
4-PHASE CLOCK GENERATOR (SHEET 1)
← JFA 0–3
BITS 7–11 →
BITS 12–15
PH1, 3, 5, AND 7
CONTROL READ ONLY MEMORY (CROM) (SHEET 2)
CONTROL BUS (4 BITS)
FLAG ENABLE
CONDITIONAL JUMP MULTIPLEXER AND CONTROL FLAG LOGIC (SHEET 1)
CONDITIONAL JUMP MULTIPLEXER (SHEET 1)
REGISTER, ARITHMETIC, AND LOGIC UNIT (RALU) NO. 1 (SHEET 2)
REGISTER, ARITHMETIC, AND LOGIC UNIT (RALU) NO. 2 (SHEET 2)
REGISTER, ARITHMETIC, AND LOGIC UNIT (RALU) NO. 3 (SHEET 2)
REGISTER, ARITHMETIC, AND LOGIC UNIT (RALU) NO. 4 (SHEET 2)
STATUS FLAG
JUMP/FLAG ADDRESS BUFFER (SHEET 2)
JUMP/FLAG ADDRESS (JFA0–3)
JUMP/FLAG ADDRESS LATCH (SHEET 1)
CONTROL FLAGS (SHEET 1)
BITS 7–11
STATUS FLAGS
PERIPHERAL DATA (16 BITS)
MEMORY DATA (16 BITS)
INPUT MULTIPLEXER (SHEET 2)
DATA BUS (16 BITS)
DATA BUFFER (SHEET 2)
BUFFERED DATA OUT (BDO) – 16 BITS
ADDRESS (16 BITS)
BITS 12–15
ADDRESS LATCHES (SHEET 2)
TIMING
HOLD
MEMORY TIMING LOGIC (SHEET 1)
TIMING
HOLD
INSTRUCTION BITS 12–15
INSTRUCTION BITS 7–11
MEMORY DATA OUT (MDO) – 16 BITS

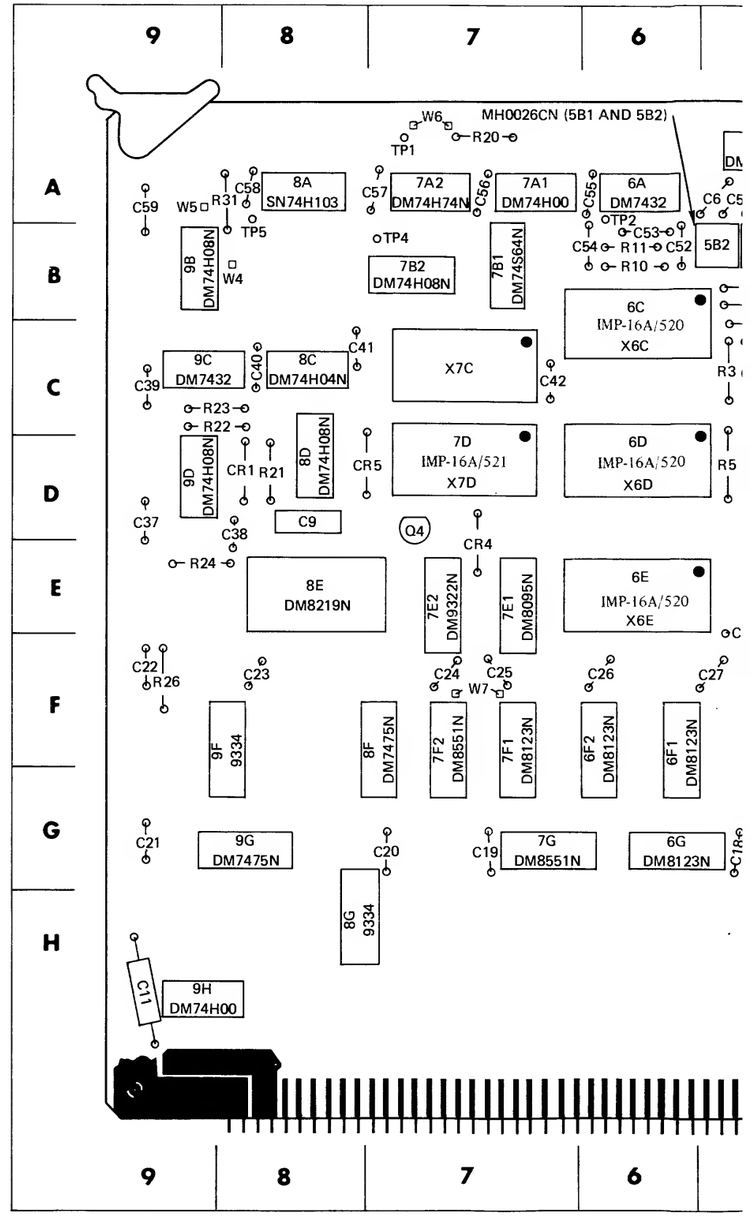


NS00137

Figure 4–7. IMP-16C Functional Block Diagram

Table 4–2. IMP-16C/200/300 Parts List

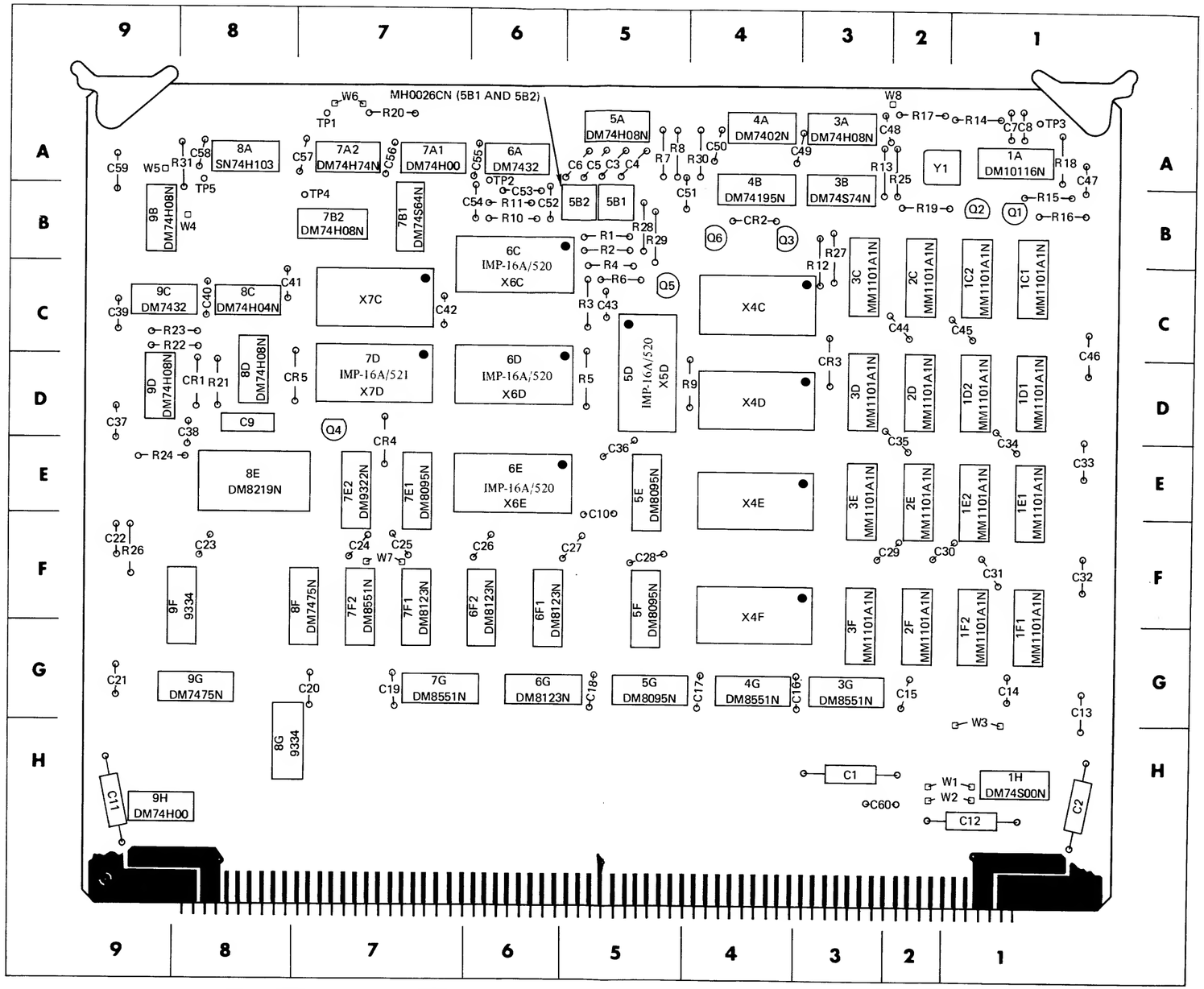
Item	Description	Reference Designation	Part Number	Quantity
	Capacitors			
1	Capacitor, 33μf, 20V	C1, C2, C11, C12		4
2	Capacitor, 300 pf, 300V	C3, C4, C6		3
3	Capacitor, 0.01μf, 50V	C5, C43		2
4	Capacitor, 0.1μf, 50V	C7, C10, C13–C42, C44–C60		49
5	Capacitor, 27 pf, 500V	C8		1
6	Capacitor, 150 μf, 20V	C9		1
	Crystal			
7	Crystal, 5.7142 MHz	Y1		1
	Diodes			
8	Diode	CR1	1N645	1
9	Diode	CR2–CR5	1N4454	4
	Resistors			
10	Resistor, 15 ohms, ±5%, 1/4W	R1–R4		4
11	Resistor, 10K, ±5%, 1/4W	R5–R8, R23		5
12	Resistor, 5.1K, ±5%, 1/4W	R9–R11, R27		4
13	Resistor, 1K, ±5%, 1/4W	R12, R20, R28–R30, R26		6
14	Resistor, 330 ohms, ±5%, 1/4W	R13, R18		2
15	Resistor, 220 ohms, ±5%, 1/4W	R14, R19, R31		3
16	Resistor, 100 ohms, ±5%, 1/4W	R15, R17		2
17	Resistor, 39 ohms, ±5%, 1/4W	R16		1
18	Resistor, 300 ohms, ±5%, 1/4W	R21		1
19	Resistor, 120 ohms, ±5%, 1/4W	R22		1
20	Resistor, 2K, ±5%, 1/4W	R24		1
21	Resistor, 2.2K, ±5%, 1/4W	R25		1
	Transistors			
22	Transistor	Q1, Q2	2N4258A	2
23	Transistor	Q3, Q4	2N4275	2
24	Transistor	Q5	2N2222A	1
25	Transistor	Q6	2N3638	1
	Integrated Circuits			
26	MOS/LSI Register, Arithmetic, and Logic Unit (RALU)	5D, 6C, 6D, 6E	IMP-16A/520	4
27	MOS/LSI Control Read-Only Memory (CROM)	7D	IMP-16A/521	1
28	MOS/LSI Control Read-Only Memory (CROM) – Optional	7C	IMP-16A/522	1
29	256-Bit Static Random Access Memory	1C1, 1C2, 2C, 3C, 1D1, 1D2, 2D, 3D, 2E, 3E, 1E1, 1E2, 2F, 3F, 1F1, 1F2	MM1101A1	16
30	Electrically Programmable 2048-Bit Read-Only Memory (PROM) – Optional	4C, 4D, 4E, 4F	MM5203	4
31	TRI-STATE®16-Line to 1-Line Multiplexer	8E	DM8219	1
32	Schottky – Clamped Transistor-Transistor Logic AND-OR-INVERT	7B1	DM74S64	1
33	Schottky – Clamped Transistor-Transistor Logic Dual D Flip-Flop	3B	DM74S74	1
34	Schottky – Clamped Transistor-Transistor Logic Quad NAND Gate	1H	DM74S00	1
35	TRI-STATE® Quad D Flip-Flop	3G, 4G, 7G, 7F2	DM8551	4
36	Quad Latch	8F, 9G	DM7475	1
37	Quad 2-Input Multiplexer	7E2	DM9322	1
38	Dual D Edge-Triggered Flip-Flop	7A2	DM74H74	1
39	4-Bit Parallel-Access Shift Register	4B	DM74195	1
40	8-Bit Addressable Latch	8G, 9F	DM9334	2
41	Dual JK Edge-Triggered Flip-Flop	8A	SNH103	1
42	Quad 2-Input NAND Gate	7A1, 9H	DM74H00	2
43	Quad 2-Input OR Gate	6A, 9C	DM7432	2
44	Quad 2-Input NOR Gate	4A	DM7402	1
45	Quad 2-Input AND Gate	3A, 5A, 7B2, 8D, 9B, 9D	DM74H08	6
46	Hex Inverter	8C	DM74H04	1
47	Hex Buffer	5E, 5F, 5G, 7E1	DM8095	4
48	TRI-STATE® Quad 2-Input Multiplexer	6G, 6F1, 6F2, 7F1	DM8123	4
49	Triple Differential Line Receiver	1A	DM10116	1
50	5 MHz 2-Phase MOS Clock Driver	5B1, 5B2	MH0026CN	2



NOTES:
(1) FOR DEVICES THAT ARE REMOVABLE, A LARGE DOT INDICATES THE LOCATION O
(2) ODD-NUMBERED PINS ARE LOCATED ON COMPONENT SIDE.
(3) EVEN-NUMBERED PINS ARE LOCATED ON SOLDER SIDE.
(4) PREFIX X DENOTES SOCKET.
(5) LOCATIONS X4C, X4D, X4E, X4F AND X7C HAVE SOCKETS ONLY; THESE COMPONENTS TO BE INSERTED BY CUSTOMER.

Table 4–2. IMP-16C/200/300 Parts List

Description	Reference Designation	Part Number	Quantity
33µf, 20V	C1, C2, C11, C12		4
300 pf, 300V	C3, C4, C6		3
0.01µf, 50V	C5, C43		2
0.1µf, 50V	C7, C10, C13–C42, C44–C60		49
27 pf, 500V	C8		1
150 µf, 20V	C9		1
142 MHz	Y1		1
	CR1	1N645	1
	CR2–CR5	1N4454	4
ohms, ±5%, 1/4W	R1–R4		4
K, ±5%, 1/4W	R5–R8, R23		5
1K, ±5%, 1/4W	R9–R11, R27		4
, ±5%, 1/4W	R12, R20, R28–R30, R26		6
0 ohms, ±5%, 1/4W	R13, R18		2
0 ohms, ±5%, 1/4W	R14, R19, R31		3
0 ohms, ±5%, 1/4W	R15, R17		2
ohms, ±5%, 1/4W	R16		1
0 ohms, ±5%, 1/4W	R21		1
0 ohms, ±5%, 1/4W	R22		1
, ±5%, 1/4W	R24		1
2K, ±5%, 1/4W	R25		1
	Q1, Q2	2N4258A	2
	Q3, Q4	2N4275	2
	Q5	2N2222A	1
	Q6	2N3638	1
uits			
egister, Arithmetic, and Logic Unit (RALU)	5D, 6C, 6D, 6E	IMP-16A/520	4
ntrol Read-Only Memory (CROM)	7D	IMP-16A/521	1
ntrol Read-Only Memory (CROM) – Optional	7C	IMP-16A/522	1
ic Random Access Memory	1C1, 1C2, 2C, 3C, 1D1, 1D2, 2D, 3D, 2E, 3E, 1E1, 1E2, 2F, 3F, 1F1, 1F2	MM1101A1	16
Programmable 2048-Bit Read-Only Memory (PROM) – Optional	4C, 4D, 4E, 4F	MM5203	4
®16-Line to 1-Line Multiplexer	8E	DM8219	1
Clamped Transistor-Transistor Logic AND-OR-INVERT	7B1	DM74S64	1
Clamped Transistor-Transistor Logic Dual D Flip-Flop	3B	DM74S74	1
Clamped Transistor-Transistor Logic Quad NAND Gate	1H	DM74S00	1
® Quad D Flip-Flop	3G, 4G, 7G, 7F2	DM8551	4
	8F, 9G	DM7475	1
t Multiplexer	7E2	DM9322	1
-Triggered Flip-Flop	7A2	DM74H74	1
-Access Shift Register	4B	DM74195	1
sable Latch	8G, 9F	DM9334	2
e-Triggered Flip-Flop	8A	SNH103	1
t NAND Gate	7A1, 9H	DM74H00	2
t OR Gate	6A, 9C	DM7432	2
t NOR Gate	4A	DM7402	1
t AND Gate	3A, 5A, 7B2, 8D, 9B, 9D	DM74H08	6
	8C	DM74H04	1
® Quad 2-Input Multiplexer	5E, 5F, 5G, 7E1	DM8095	4
ential Line Receiver	6G, 6F1, 6F2, 7F1	DM8123	4
se MOS Clock Driver	1A	DM10116	1
	5B1, 5B2	MH0026CN	2



NOTES:
(1) FOR DEVICES THAT ARE REMOVABLE, A LARGE DOT INDICATES THE LOCATION OF PIN 1.
(2) ODD-NUMBERED PINS ARE LOCATED ON COMPONENT SIDE.
(3) EVEN-NUMBERED PINS ARE LOCATED ON SOLDER SIDE.
(4) PREFIX X DENOTES SOCKET.
(5) LOCATIONS X4C, X4D, X4E, X4F AND X7C HAVE SOCKETS ONLY; THESE COMPONENTS TO BE INSERTED BY CUSTOMER.

INFORMATION ON THIS DIAGRAM IS SUBJECT TO CHANGE WITHOUT NOTICE. ENGINEERING DIAGRAMS SUPPLIED WITH IMP-16C SHOULD BE REFERENCED.

Figure 4–8. IMP-16C Card, Component Layout

Chapter 5
INPUT/OUTPUT OPERATIONS

5.1 INPUT/OUTPUT INSTRUCTIONS

Input/output operations are carried out with the RIN (Register In) and ROUT (Register Out) macroinstructions. Functionally, they are similar to the LOAD and STORE instructions in that they address a particular device and initiate data exchanges. The effective address of an input/output device is determined by the sum of the contents of Accumulator 3 (AC3) and the 7-bit control field of the RIN or ROUT instruction. Timing for these instructions is shown in figure 5–2.

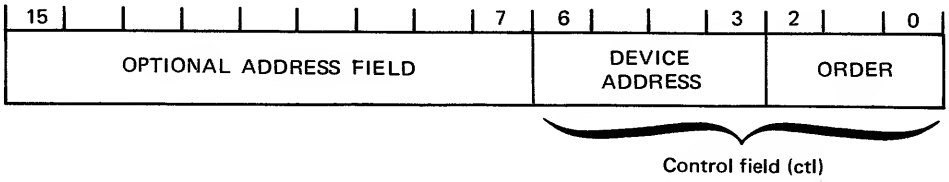


Figure 5–1. Input/Output Word Format

Sixteen bits are available for address and command codes. Although a number of schemes are possible, the one described here has proved useful for many applications. The low-order 3 bits may be used to define an input/output "order," and bits 3 to 6 are the device addresses (figure 5–1). Each peripheral device decodes the address field of the input/output instruction command, and if the Read Peripheral or the Write Peripheral flag is active, the device will respond. The 3 "order" bits permit eight possible auxiliary operations for each input/output class; for example, these orders may be read data, read status, reset device, rewind tape, backspace, write data, and so on. The assignment of the various orders is left to the systems programmer.

The 4 bits of the device address field permit direct addressing of 16 devices; however, by loading Accumulator 3 (which is added to bits 0 to 6 of the instruction) with a 16-bit value before executing a RIN or ROUT instruction, up to 65,536 addresses may be specified.

5.2 DATA TRANSFER TO PERIPHERAL DEVICES

Peripheral device data may be accessed by the Read Peripheral and Write Peripheral control flags in conjunction with the peripheral device address. Actual transfer of data is effected by the RIN and ROUT instructions during T7 (input) and T4 (output), respectively (figure 2–4). Peripheral device control may be carried out by using the order field of the instruction (figure 5–1) or by dedicating a general-purpose control flag to a peripheral device control function, which can then be controlled by use of the SFLG and PFLG instructions. Similarly, peripheral device status can be sensed by issuing an order to read status over the data bus, or by dedicating one of the general-purpose user jump conditions to this function and using the BOC instruction. (The control flags and jump condition inputs can actually be used to implement a very low-cost serial-data interface without using the data bus at all.)

The functions performed as a result of an input/output command vary. For example, a read peripheral order to a magnetic disc typically initiates a block transfer of information. In contrast, a similar order to a Teletype typically executes the transfer of a single character.

The use of input/output instructions is best illustrated by an example. Consider the case of reading in characters from a serial Teletype unit and transmitting them back immediately (echoing). The following program segment will do this effectively. The first few comment lines define the input/output orders for the Teletype unit. It is assumed that the Teletype is sending data serially over the line corresponding to bit 15.

```
Instruction                    Comment

;                              DEFINE I/O ORDERS.
;                              5 = RESET
;                              2 = READ DATA
;                              4 = ENABLE TTY PAPER TAPE READER
;                              3 = WRITE DATA
;                              TTYAD = TTY ADDRESS IN BIT 3-6
;                              C1 = 1 (BRANCH IF AC0 = 0)
START:  LI 3, TTYAD            LOAD AC3 WITH ADDRESS OF TTY.
        ROUT 5;                RESET TELETYPE (ORDER FIELD = 5).
        LI 2, 8                SET COUNT FOR 8 BITS.
READ:   ROUT 4;                ENABLE TTY READER
        RIN 2;                 READ IN TTY DATA
        BOC C2, .+2;           TEST FOR START BIT; C2 REFERS TO AC0 = 0.
        JUMP READ;             READ DATA AGAIN (UNTIL START BIT FOUND).
        JSR DELAY;             START BIT FOUND; DELAY FOR PROPER TIMING.
INPUT:  ROUT 3;                SEND BIT TO PRINTER.
        JSR DELAY;             DELAY ROUTINE TO TIME OUT 1 BIT
        RIN 2;                 READ TTY DATA.
        SHR 0, 1;              SHIFT DATA ONE POSITION.
        AISZ 2, -1;            TEST TO SEE IF DONE
        JMP INPUT;             NOT DONE; READ MORE DATA.
        .                      .
        .                      .
        .                      .
DELAY:  .;                     START OF DELAY SUBROUTINE
```

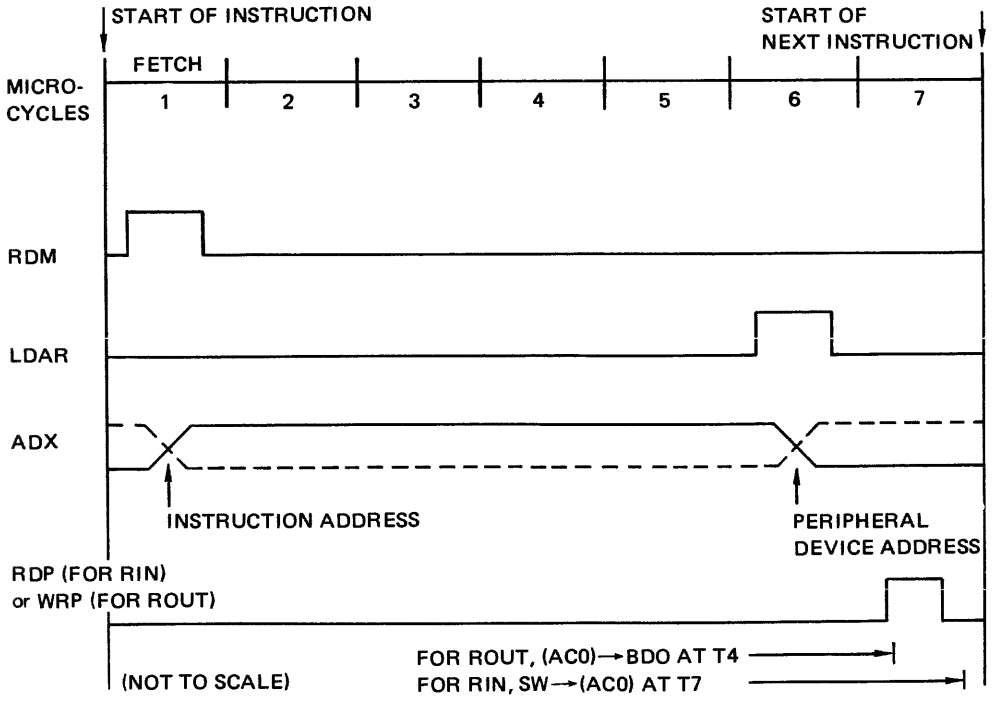


Figure 5–2. Timing Sequence for RIN and ROUT Instructions

Chapter 6
INTERRUPT SYSTEM

The IMP-16C system recognizes one level of interrupt in its present configuration. A general interrupt request is initiated by the Interrupt Request Signal (INTRA) to the Interrupt Handler (sheet 1, figure 4–6). Also provided is a control panel interrupt input (CPINT). The workings of both types are described below.

6.1 GENERAL INTERRUPT

A peripheral device (or any external condition) may send an interrupt request to the IMP-16C over the INTRA line. If the Interrupt Enable Flag is set (that is, no other interrupt currently being serviced), then the interrupt request is latched in a flip-flop and awaits service. During the next instruction fetch cycle, the processor resets the Interrupt Enable Flag (INTEN) and transfers control to location 1 in main memory. At the same time, the PC value is saved on the stack.

The instruction in location 1 of main memory typically would be the start of an interrupt service routine or a jump to a service routine. In the IMP-16C, the stack overflow condition causes an interrupt on the same line. The interrupt service routine can detect this type of interrupt by using a Branch-On Condition (BOC) instruction with cc = 8 (stack full). The interrupt sequence is best illustrated by an example.

6.2 EXAMPLE OF INTERRUPT REQUEST AND SERVICE

The case considered here is that of a real-time clock that provides interval timing by sending timed interrupts to the processor. The hardware for this feature would consist of a presettable counter that raises a status signal after it has counted through its sequence. This signal can be used as an interrupt request.

As an example of the use of this timer, consider an application where it is desired to sample a waveform at regular intervals. The real-time clock can be used to generate interrupts at these intervals, and a processing subroutine can read the contents of an analog-to-digital converter driven by the waveform under test. The following program segment shows how this can be done. The clock is assumed to use bit 0 on the data bus to signal its status. This signal is also wired to the general interrupt request line. The interrupt status of the clock is read over bit 0 of the data bus by issuing a READ STATUS order. (Note that other devices could use other bits of the data bus to respond to this order simultaneously. The data bits may then be tested to determine devices that are requesting interrupt service.)

The first few lines of the program are comments that describe the various order codes and addresses assigned to the real-time clock and the A/D converter. The remaining lines of the program segment perform operations according to the requirements above; the comments serve to annotate the program.

```
Label           Instruction          Comment

;                                    INTERRUPT SEQUENCE FOR REAL-TIME
;                                    CLOCK AND WAVEFORM SAMPLER
;                                    RTC = ADDRESS OF REAL-TIME CLOCK
;                                    ADC = ADDRESS OF A/D CONVERTER
;                                    2 = START CLOCK (I/O ORDER)
;                                    1 = READ DATA ORDER
;                                    0 = READ INTERRUPT STATUS ORDER
;                                    C3 = 3; CC FOR BIT 0 OF AC0 = 1
;                                    CLOCK INTERRUPT STATUS HAS BEEN ASSIGNED BIT 0 ON BUS
  LOC1:         JMP    INTR:       THIS INSTRUCTION IS IN LOCATION X'0001
; MAIN PROGRAM FOLLOWS:
                SFLG   1;          ENABLE INTERRUPT SYSTEM
                       :
                       :
                LI     3, RTC;     LOAD AC3 WITH CLOCK ADDRESS
                ROUT   2;          START TIMER ON CLOCK
                       :
                       :
; SERVICE ROUTINE FOLLOWS:
  INTR:         LI     3, 0;       CLEAR AC3 BEFORE EXECUTING RIN
                RIN    0;          READ DATA BUS TO CHECK INTERRUPTING DEVICE.
                BOC    C3, CLKINT; BRANCH TO CLOCK SERVICE IF BIT 0 = 1.
                       :
                       :
; CLOCK SERVICE ROUTINE FOLLOWS:
  CLKINT:       LI     3, RTC;     LOAD AC3 WITH CLOCK ADDRESS
                ROUT   2;          RESTART TIMER
                LI     3, ADC;     LOAD ADDRESS OF A/D CONVERTER
                RIN    1;          READ ADC DATA.
                JSR    SAMPLE;     GO TO DATA SAMPLE PROCESSING SUBROUTINE.
                RTI    ;           RETURN FROM INTERRUPT.
                       :
                       :
; CONTINUE PROCESSING OF OTHER INTERRUPTS.
                       :
                       :
SAMPLE: . . . . . .;               START OF ROUTINE TO PROCESS DATA FROM A/D CONVERTER.
```

6.3 CONTROL PANEL INTERRUPT

The IMP-16C microprogram allows a high-priority interrupt to be indicated on the CPINT line. Since this interrupt is useful for implementing a "program-controlled" control panel, it is called the Control Panel Interrupt; it may, of course, be used for other purposes. If CPINT is active, the processor pulses flag CPINP, reads the data bus at T7 of the same microcycle, and responds as if the data were an instruction. The CPINT input, therefore, can be used to force a jump to some reserved location by "jamming" an instruction on the data lines. This may be used to cause a branch to a control panel service routine or to provide vectored interrupts. By forcing the bus to send in all zeros, the CPINT feature may be used to incorporate a single-step feature, by causing the CPU to halt after the execution of one instruction.

6.4 MULTILEVEL INTERRUPTS

It is possible to use the IMP-16C for multilevel interrupt service by use of the RALU general-purpose status flags. Two of these (FLAG0 and FLAG12) are available at the card-edge connectors for this purpose (or other applications where it is desirable to save status). These flags may be used as interrupt enable flags for two individual levels; they can be modified through use of the PUSHF, PULLF, PULL, and PUSH instructions. A typical arrangement that makes use of the available INTRA input is shown in figure 6–1. (The general-purpose control flags could also be used for this purpose, but their state cannot readily be preserved on the stack as can the status flags.)

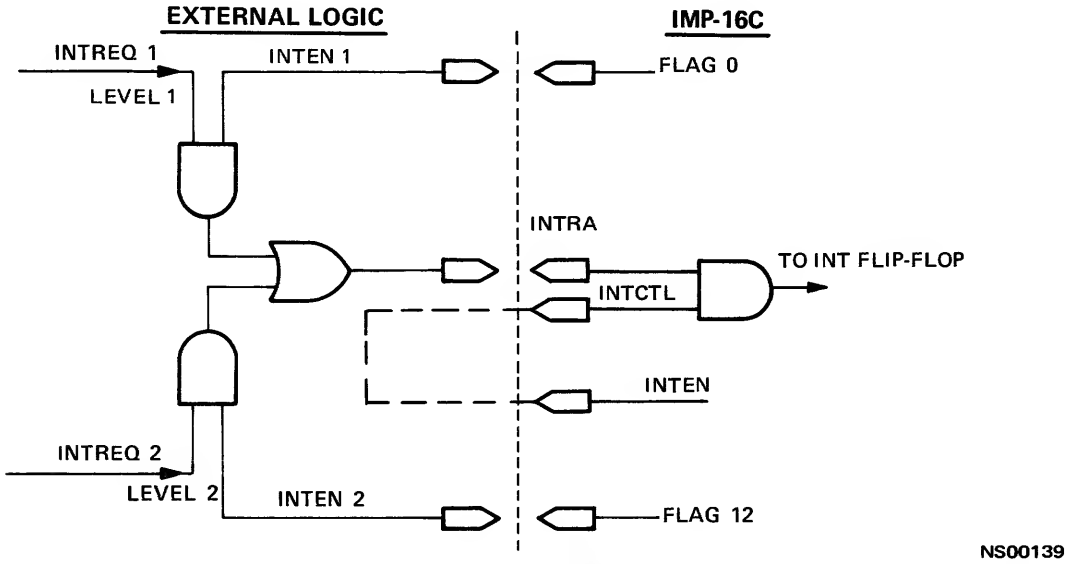


Figure 6–1. Use of INTRA Input

The processor responds to inputs on the interrupt request lines that are labeled INTREQ1 and INTREQ2 in figure 6–1. Several devices may be "wire-ORed" to each of these interrupt request lines. If any device generates an interrupt request, the line will go high and interrupt the processor if that level of interrupt is enabled. There is a separate interrupt enable flag for each of the two interrupt levels (labeled INTEN1 and INTEN2) and a master interrupt enable (labeled INTEN) for all levels plus the stack overflow interrupt. The interrupt enable flags for each individual level are part of the CPU status flags, and, as mentioned above, they can be modified by use of the PUSHF, PULLF, PULL, and PUSH instructions. The INTEN flag is one of the IMP-16C control flags. It is modified by use of the Set Flag (SFLG) and the Pulse Flag (PFLG) instructions.

The availability of several different interrupt levels provides a convenient means of controlling interrupts for devices of different priority in a system with a number of peripherals. When only one interrupt request line is used, the interrupt service program must issue an order to each device of lower priority than the one being serviced to reset the lower priority interrupt enable flags on the peripheral controllers. This could be very time consuming in a system with many peripherals. In a system with several interrupt levels, all devices of like priority are tied to the same interrupt request line. For all devices on an individual level that have a common interrupt enable flag at the processor, the interrupt enable flag can disable all devices on that level simultaneously. When an interrupt occurs, lower priority levels are therefore disabled in a minimum amount of time.

The interrupt handling process is summarized by the following. When the processor responds to an interrupt request, it performs the following tasks in order to transfer program control to the interrupt service routine:

- Checks the type of interrupt; if caused by CPINT, jumps to CPINT service routine.
- Transfers the contents of the Program Counter (PC) to the top of the stack.
- Places the address of memory location 1 into the PC.

- Disables (clears) the processor's Interrupt Enable (INTEN) flag to prevent further interrupt.
- Fetches and executes the next instruction from memory location 1, thus initiating the interrupt service routine.

The CPINT service routine does the following:

- Pulses the CPINP flag, thereby indicating that a control panel interrupt is active.
- Reads the data bus and treats the incoming data as an instruction to be executed.

Chapter 7
CONTROL PANEL AND 4K-BY-16 MEMORY

The IMP-16C is self-contained for controller applications. However, for some general-purpose applications, a control panel may be needed or convenient to have for debugging, verifying operation, troubleshooting, and other operator-controlled uses. Also, additional memory capacity may be required, and this is available in the National Semiconductor 4K-by-16 random access memory modules.

7.1 CONTROL PANEL OPERATION

The simplest control panel for use with the IMP-16C is one that is serviced by software and has a minimum of extra components. Such a panel would consist of a set of 16 data switches, 16 indicator lights, and a few active function switches. These are represented schematically in figure 7–1. The Light-Emitting Diode (LED) devices are driven by a set of DM7475 latches that are strobed by a WRITE PERIPHERAL flag pulse; in this mode, the lights are considered to be an output device.

The spare jump condition multiplexer inputs on the IMP-16C may be tied to active switches on the control panel. These switches may be used for controlling operations such as Load Address, Load Data, Execute, Display Data, and others.

A software service routine that resides in main memory can scan these switches using the Branch-On Condition (BOC) instruction. If any switch is active, an appropriate service action may be initiated. The following program segment is an example of such a routine. It assumes that jump conditions 12, 13, and 7 are wired to switches that cause Load Address, Load Data, and Execute operations to occur. This simple example assumes that there is no peripheral address decoding; all input/output operations are activated by READ PERIPHERAL and WRITE PERIPHERAL control flags as commanded by RIN and ROUT instructions.

```
C12 = 12          ;     LOAD ADDRESS CONDITION.
C13 = 13          ;     LOAD DATA CONDITION.
C7  = 7           ;     EXECUTE
 =X'FFFE          ;     ASSEMBLER DIRECTIVE TO PLACE NEXT INSTRUCTION
                  ;       AT LOCATION X'FFFE.
JMP BEGIN
.
.
.
```

Program continues on page 7–3.

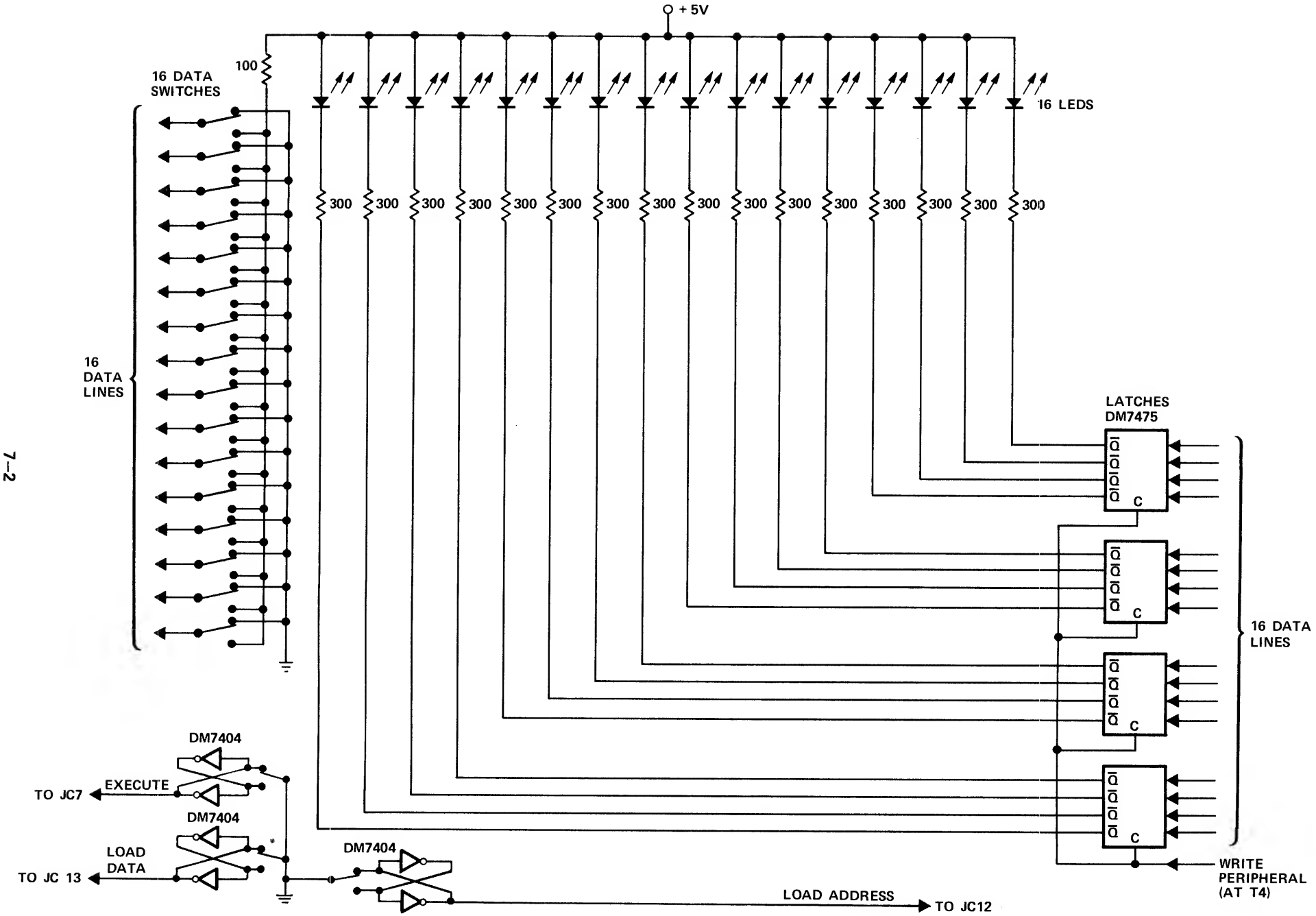


Figure 7–1. Control Panel Example

```
BEGIN:  BOC   C12, LA   ;   WAIT LOOP TO
        BOC   C13, LD   ;   SCAN PANEL
        BOC   C7, EX    ;   FUNCTION SWITCHES.
        JMP   BEGIN     ;
LA:     BOC   C12, LA   ;   SWITCH "DEBOUNCER".
        RIN             ;   READ SWITCHES.
        RCPY  0, 2      ;   SAVE ADDRESS IN AC2.
OUT:    ROUT            ;   DISPLAY ADDRESS.
        JMP   BEGIN     ;   RETURN TO WAIT LOOP.
LD:     BOC   C13, LD   ;
        RIN             ;   READ SWITCHES.
        ST    0, (2)    ;   STORE DATA IN ADDRESSED LOCATION.
        ROUT            ;   DISPLAY DATA.
        AISZ  2, 1      ;   INCREMENT ADDRESS.
        JMP   BEGIN     ;   RETURN TO WAIT LOOP.
        HALT            ;   END OF ADDRESS RANGE.
EX:     BOC   C7, EX    ;
        RIN             ;   READ SWITCHES.
        RCPY  0, 2      ;
        JMP   (2)       ;   JUMP TO STARTING ADDRESS.
        .
        .
        .
```

The program above is an example of how the jump condition inputs can be used to scan external conditions and/or requests.

7.2 DYNAMIC MEMORY INTERFACE

A circuit has been provided on the IMP-16C card to enable interfacing with the National Semiconductor 4K-by-16 dynamic memory modules. This circuit consists of logic that can receive memory refresh request and send out appropriate refresh orders and read/write cycle initiate signals. The refresh logic appears on the IMP-16C schematic diagram (figure 4–6) but is repeated in simplified form in figure 7–2.

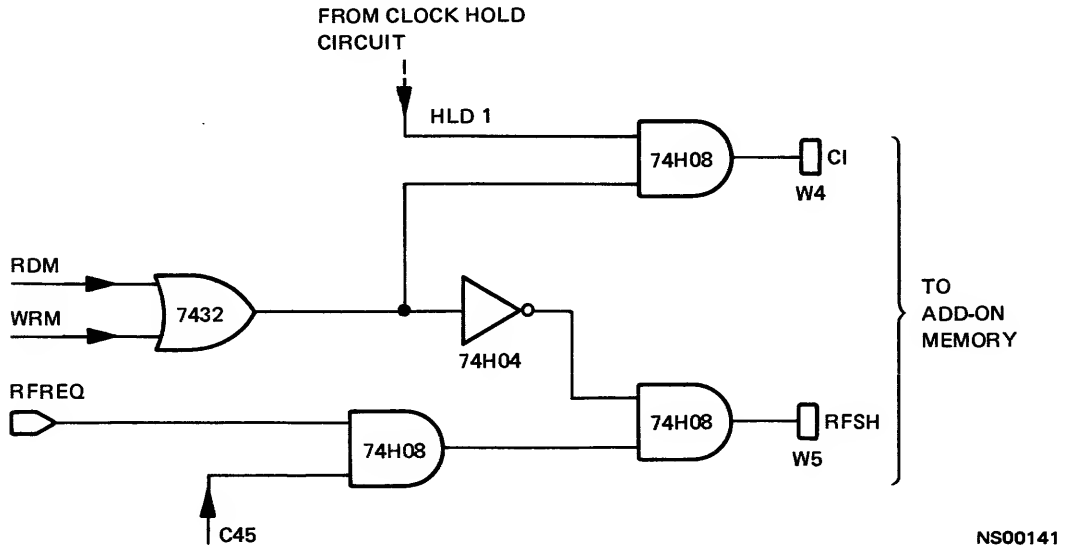
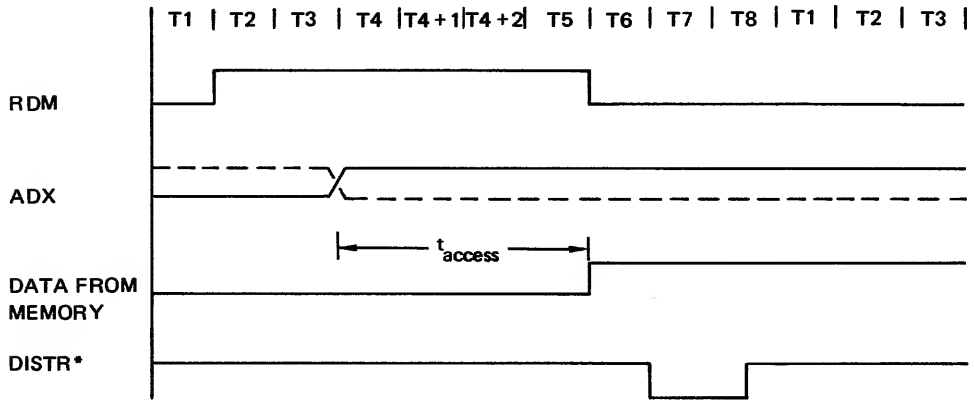


Figure 7–2. Refresh Logic for Dynamic Memory

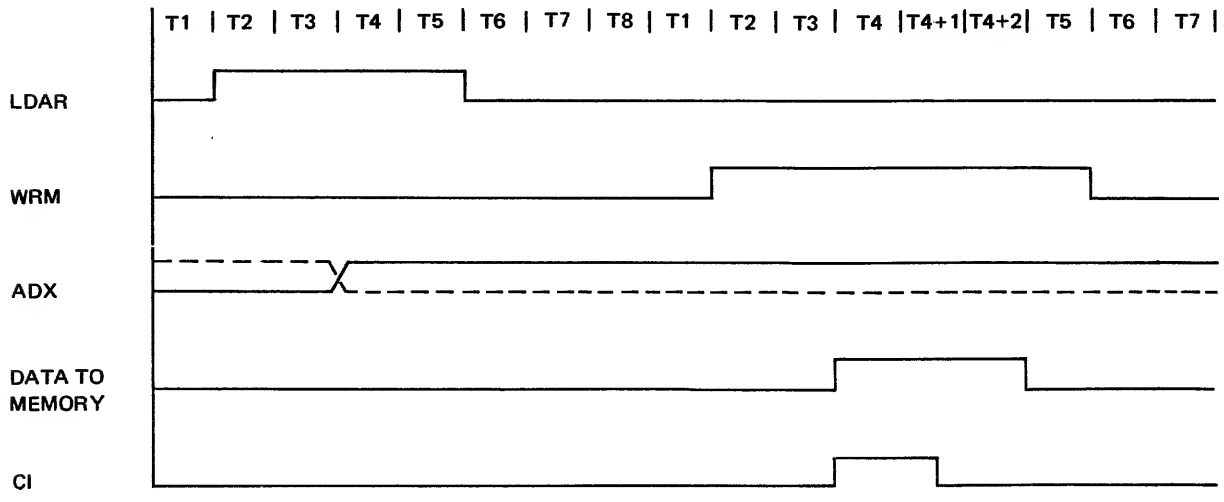
The 4K-by-16 memory requires a refresh cycle of 1 μs duration every 60 μs. The IMP-16C makes a memory reference during every fetch operation and a few other times in-between, depending on the instruction being executed. If the processor is busy during a read or write operation, the memory waits for a refresh cycle until the processor is free. The refresh request signal (RFREQ) is gated into the circuitry at T4. If, at this time, a memory read/write is not requested, an RFSH signal is sent to the memory; otherwise, a Memory Cycle Initiate (CI) signal is sent to memory. Timing for memory read and write operations is given in figure 7–3.

The memory option is provided in the form of a jumper-connectable circuit. If used, jumper pads W4 and W5 may be connected to any two of the unused pins on the edge connector of the IMP-16C.

If add-on memory is attached to the IMP-16C, the user must be aware that the added capacitive loading will slow down the memory access time. Typically, if more than 8K of additional memory is used, the clock periods must be stretched to provide extra time. This may be done using the external hold input which is described in chapter 9.



(a) READ CYCLE TIMING



(b) WRITE CYCLE TIMING

Figure 7–3. Memory Timing Waveforms

Chapter 8
SYSTEM VERIFICATION

8.1 INTRODUCTION

The IMP-16C is a large-scale integrated (LSI) processor consisting of several components from different logic families and types. As such, the proper functioning of the IMP-16C is dependent on the interfacing and timing between the various components. In order to facilitate hardware debugging, a brief system verification procedure is described in this chapter to aid users in the understanding of the system.

For system verification, the IMP-16C may be divided into four distinct partitions:

- Timing and Clocks
- MOS CPU Logic
- Control Flags and Logic
- Data Buses

8.2 TIMING AND CLOCKS

When the system is started with the initialization sequence described in chapter 4, the system initialize signal (INIT*) enables all the timing signals. The clocks start with phase 1 and generate all the other auxiliary timing signals. Figure 8–1 shows all the clock signals and their timing relationships. After it has been verified that the MOS clocks are being generated properly, the MOS CPU logic can be checked out.

8.3 MOS CPU LOGIC

If the system is functioning properly, the CPU executes instructions stored in external memory or executes an instruction(s) from a switch register. In either case, if instructions are being executed, the CROM cycles through repeated FETCH operations. By monitoring the CROM ENCTL line (see sheet 2, figure 4–6), this fact can be verified. The ENCTL line is high between T2 and T7 of every cycle immediately preceding a fetch (except for multiple shift/rotate operations). If the number of microcycles between successive ENCTL pulses is counted for each instruction and checked against table A–1 in appendix A, then it can be assumed that at least most of the CROM circuits are functioning (see figure 8–2).

If the system is being used with some form of control panel (as, for example, the one described in chapter 7), then the RALU can be checked out by outputting register values to a set of display lights.

8.4 CONTROL FLAGS AND LOGIC

Another convenient test point for system verification is the Read Memory (RDM) flag. This flag is pulsed between T2 and T6 during every fetch operation. If operation of the CROM has been verified, then this flag should come up properly (figure 8–2).

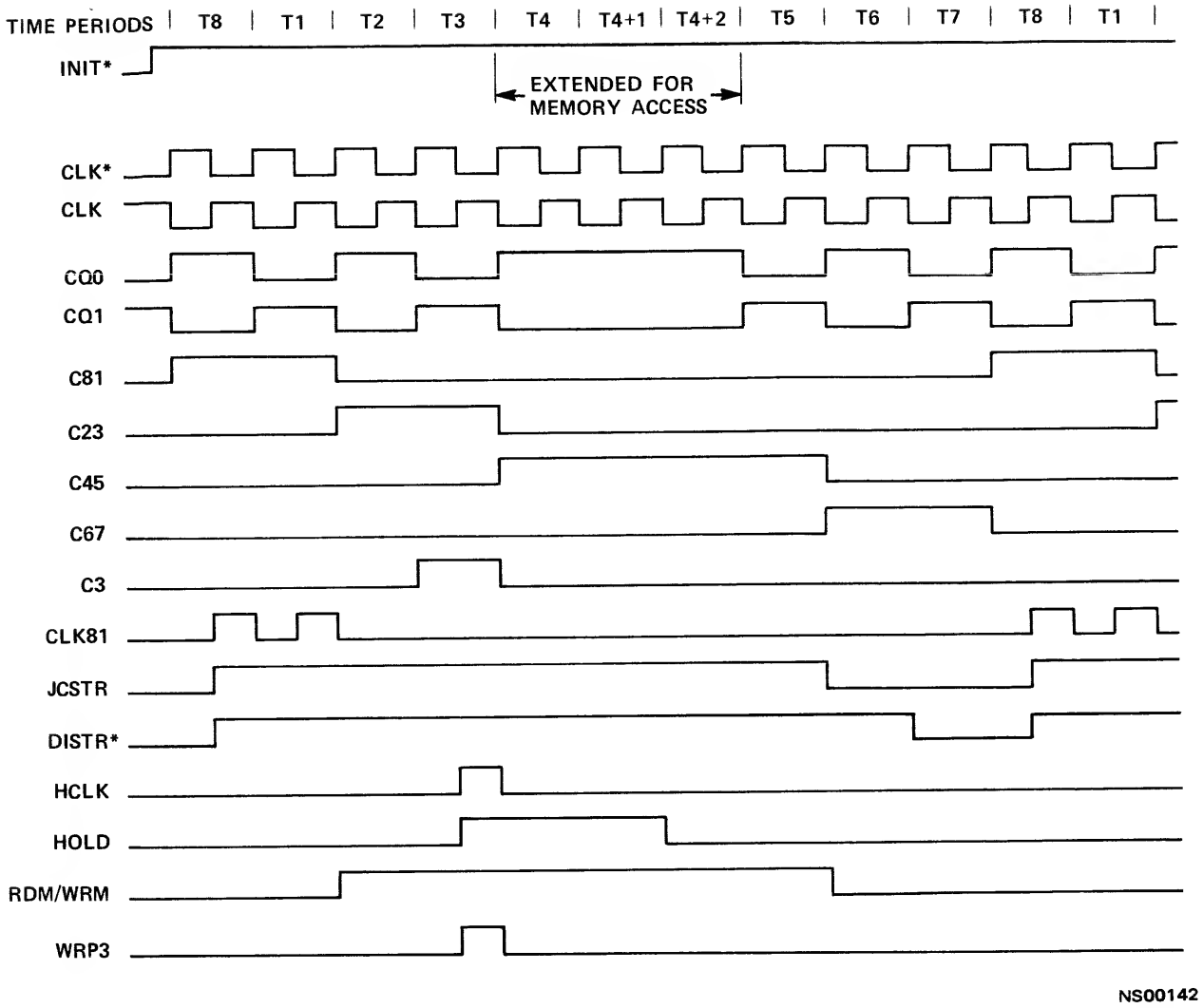


Figure 8–1. IMP-16C Clocks

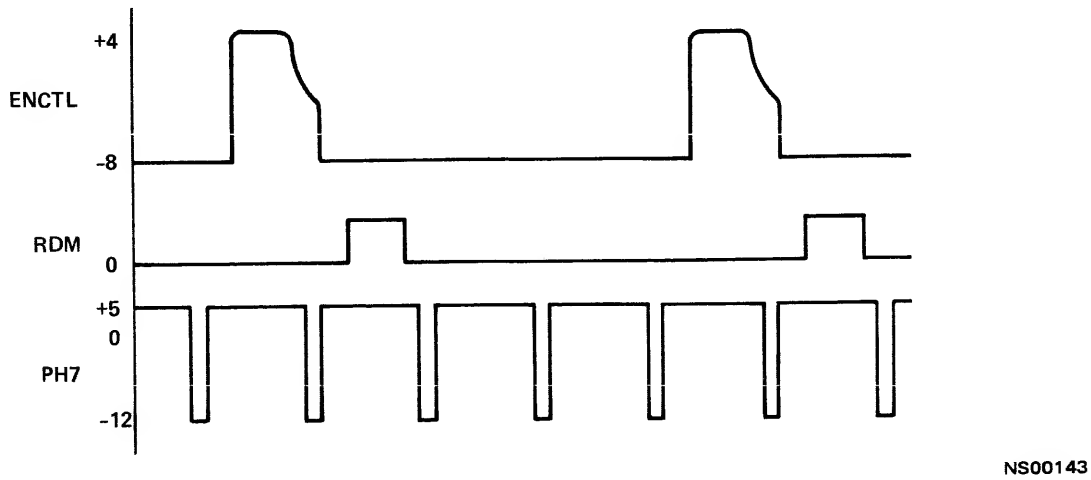


Figure 8–2. Microcycle Timing Sequence for JSR Instruction

8.5 DATA BUSES

The data buses can be checked out by the following:

- Reading in some switch values and seeing if they appear on the bus data output lines.
- Checking the address register to see if the data bus value is being latched during T4 when the RDM flag is up.

8.6 DIAGNOSTIC PROGRAMS

The IMP-16C card may be checked out functionally with the use of special diagnostic programs stored in four electrically programmable ROMs (or PROMs). They may be used for system verification by inserting them in the read-only memory sockets in the card and observing the various status indications provided by the program. The diagnostic routines are designed to exercise all parts of the CPU for proper operation. These programs are available as an option (IMP-16F/501).

8.7 OPERATING PROCEDURES

For the proper functioning of the IMP-16C in a system environment, the following points should be observed carefully.

(1) If any of the following signal lines to the IMP-16C is not used, it should be tied to a logic "0" (ground) level:

  (a) Any unused jump conditions (JC12 through JC15; EXEC)
  (b) General interrupt line (INTRA)
  (c) Control Panel interrupt line (CPINT)

(2) If the output disable feature of the address bus is not used, the ODIS line should be tied to a logic "0" (ground) level.

(3) An interrupt control input (INTCTL) is provided for externally enabling the interrupt system. For normal operation, this line should be tied to the interrupt enable (INTEN) provided by the CPU. If the system environment is such that the interrupt system is not used at all, the INTCTL input should be tied to a logic "0."

(4) If slow memories are used (access times in excess of 850 ns), additional clock stretching must be provided by the user (see chapter 9).

(5) On the IMP-16C/200 and 300 cards, the circuit that supplies the switched −12 volts (SVGG) is part of the card itself; therefore, no external −12-volt SVGG should be connected to pins 11 and 12 of the card.

Chapter 9
USER OPTIONS

Several user-defined options are available on the basic IMP-16C card. These options pertain to system timing, bus selection, and memory management; these are implemented by removable jumper connections on the IMP-16C/200 and IMP-16C/300 cards.

9.1 EXTERNAL CLOCK HOLD

An external input (EXHOLD) is provided so the user may supply his own clock hold signal. If this input is used, jumper W6 should be removed. When using this feature, care must be taken to synchronize the external signal with the timing of the processor. This is done by making sure that the generated hold signal comes up in the middle of the time slot preceding the phase being stretched. Similarly, at the end of the hold period, the signal must be released at least 50 ns before the start of the next time slot. See figure 8–1 for this type of timing relationship.

9.2 INPUT BUS SELECTION

The input bus structure of the IMP-16C is arranged such that memory data and peripheral data are multiplexed under control of the delayed read-memory flag (RDM–Q1). It is possible to change this multiplexing operation by disabling the RDM–Q1 signal (removing jumper W7) and supplying an externally generated signal on the DSLCT input. The user may wish to do this in case his system requirements call for a common input bus for peripheral devices and any add-on memory.

9.3 ADDRESS-BUS DISABLE

The address bus on the IMP-16C (ADX00 through ADX15) may be placed in a TRI-STATE mode by taking the ODIS input to a logic "1" level. This feature allows the forcing of external addresses on the bus without CPU intervention. During normal operation, the ODIS line is at a logic "0" level.

9.4 MEMORY MANAGEMENT

As mentioned in chapter 4 (section 4.5), the on-board memory is controlled by address bit 15. If it is desired to change this method of decoding or to add on additional memory, the user may remove jumpers W1, W2, and W3 and supply externally generated decoding via input lines CS0, CS1, and CS2. To illustrate this, two examples are given below.

Example 1: If it is desired to add on more read-write memory to make a total of 4K words in the address space 0–4K, then remove jumper W1 and provide a signal at CS0 that is a logic "0" when (and only when) ADX08–ADX15 are all logic "0." This will enable the first 256-word block of memory that is on the IMP-16C card.

Example 2: If, for example, 36K of memory were required, then remove W1, W2, and W3. CS0 would be generated exactly as in example 1. CS1 would have to be a logic "0" when ADX09–ADX15 are all logic "1" with ADX08 a logic "0," and CS2 would have to be a logic "0" when ADX08–ADX15 are all logic "1."

9.5 MEMORY ACCESS TIME

The addition of extra memory to the IMP-16C will increase the capacitive loading on the system, so appropriate allowances must be made for access time degradation. The clock-hold circuit on the IMP-16C is designed to permit access times up to 850 ns. For longer access times, the external clock-hold option should be used (see 9.1).

Appendix A
SUMMARY OF INSTRUCTIONS

Table A–1. IMP-16C Basic Instruction Set (Executed by CROM I)

Instruction	Mnemonic	Execution Cycles	Memory Read Cycles	Memory Write Cycles
Memory Reference Instructions				
Load	LD	5	2	–
Load Indirect[1]	LD	5	3	–
Store	ST	6	1	1
Store Indirect[1]	ST	8	2	1
Add	ADD	5	2	–
Subtract	SUB	5	2	–
Jump	JMP	3	1	–
Jump Indirect[1]	JMP	5	2	–
Jump to Subroutine	JSR	4	1	–
Jump to Subroutine Indirect[1]	JSR	6	2	–
Increment and Skip if Zero	ISZ	7,8 if SKIP	2	1
Decrement and Skip if Zero	DSZ	8,9 if SKIP	2	1
Skip if AND is Zero	SKAZ	6,7 if SKIP	2	–
Skip if Greater	SKG	Like Signs: 8,9 if SKIP Unlike Signs: 9,10 if SKIP	2	–
Skip if Not Equal	SKNE	6	2	–
And	AND	5	2	–
Or	OR	5	2	–
Register Reference Instructions				
Push on to Stack Register	PUSH	3	1	–
Pull from Stack	PULL	3	1	–
Add Immediate, Skip if Zero	AISZ	4,5 if SKIP	1	–
Load Immediate	LI	3	1	–
Complement and Add Immediate	CAI	3	1	–
Register Copy	RCPY	6	1	–
Exchange Register and Top of Stack	XCHRS	5	1	–
Exchange Registers	RXCH	8	1	–
Register And	RAND	6	1	–
Register Exclusive Or	RXOR	6	1	–
Register Add	RADD	3	1	–
Shift Left	SHL	4 + 3K	1	–
Shift Right	SHR	4 + 3K	1	–
Rotate Left	ROL	4 + 3K	1	–
Rotate Right	ROR	4 + 3K	1	–

1 - The symbol @ must precede the designation of the memory location whose contents become the effective address by indirection.

Table A–1. IMP-16C Basic Instruction Set (Continued)

Instruction	Mnemonic	Execution Cycles	Memory Read Cycles	Memory Write Cycles
Input/Output, Flag, and Halt Instructions				
Set Flag	SFLG	4	1	–
Pulse Flag	PFLG	4	1	–
Push Flags on Stack	PUSHF	4	1	–
Pull Flags from Stack	PULLF	5	1	–
Register In	RIN	7	1	–
Register Out	ROUT	7	1	–
Halt	HALT	–	–	–
Transfer of Control Instructions				
Branch-On Condition	BOC	4,5 if branch	1	–
Return from Subroutine	RTS	4	1	–
Return from Interrupt	RTI	5	1	–
Jump to Subroutine Implied	JSRI	4	1	–

Execution Time = (E + 0.25R + 0.25W) T

where

E = number of execution cycles

R = number of memory read cycles

W = number of memory write cycles

T = time for one microcycle

The 0.25 factor for the read and write cycles is included because the main clocks are stopped for two periods during read and write operations with memory. If the clock-stop feature is not used, then there is no overhead for the read/write operations; hence 0.25 may be replaced by 0. For the shift and rotate instructions, K refers to the number of positions shifted or rotated.

As an example of the use of the formula, let T = 1.4 microseconds; then, a load-instruction execution would take

(5 + 0.50) 1.4 = 7.7 microseconds

A store-instruction execution would take

(6 + 0.25 + 0.25) 1.4 = 9.09 microseconds

Table A–2. IMP-16C Extended Instruction Set (Executed by CROM II)

Instruction	Mnemonic	Execution Cycles	Memory Read Cycles	Memory Write Cycles
Multiply	MPY	106 to 122	3	–
Divide	DIV	125 to 159	3	–
Double Precision Add	DADD	12	4	–
Double Precision Subtract	DSUB	12	4	–
Load Byte	LDB	20 (left) 12 (right)	4	–
Store Byte	STB	24 (left) 17 (right)	4	1
Set Status Flag	SETST	17 to 36	1	–
Clear Status Flag	CLRST	17 to 36	1	–
Skip If Status Flag True	SKSTF	19 to 39	1	–
Set Bit	SETBIT	15 to 34	1	–
Clear Bit	CLRBIT	15 to 34	1	–
Complement Bit	CMPBIT	15 to 34	1	–
Skip If Bit True	SKBIT	19 to 39	1	–
Interrupt Scan	ISCAN	9 to 80	1	–
Jump Indirect to Level Zero Interrupt	JINT	7	2	–
Jump Through Pointer	JMPP	7	3	–
Jump to Subroutine Through Pointer	JSRP	8	3	–

SUPPLEMENT 1 TO PUB. NO. 4200021C

Integrated MicroProcessor-16C

IMP-16C

INTERFACING GUIDE

January 1974

2900 Semiconductor Drive
Santa Clara, California 95051

PREFACE

This application note supplements the IMP-16C Application Manual and supplies information pertaining to the implementation of peripheral interfacing for the IMP-16C. The user should be familiar with the contents of the IMP-16C Application Manual, particularly with respect to the macroinstruction and timing-signal mnemonics.

The material supplied in this application note is for information purposes only and is subject to change without notice. This applies particularly to the circuit diagrams and the computer program listings.

CONTENTS

CONTENTS (Continued)

ILLUSTRATIONS

TABLES

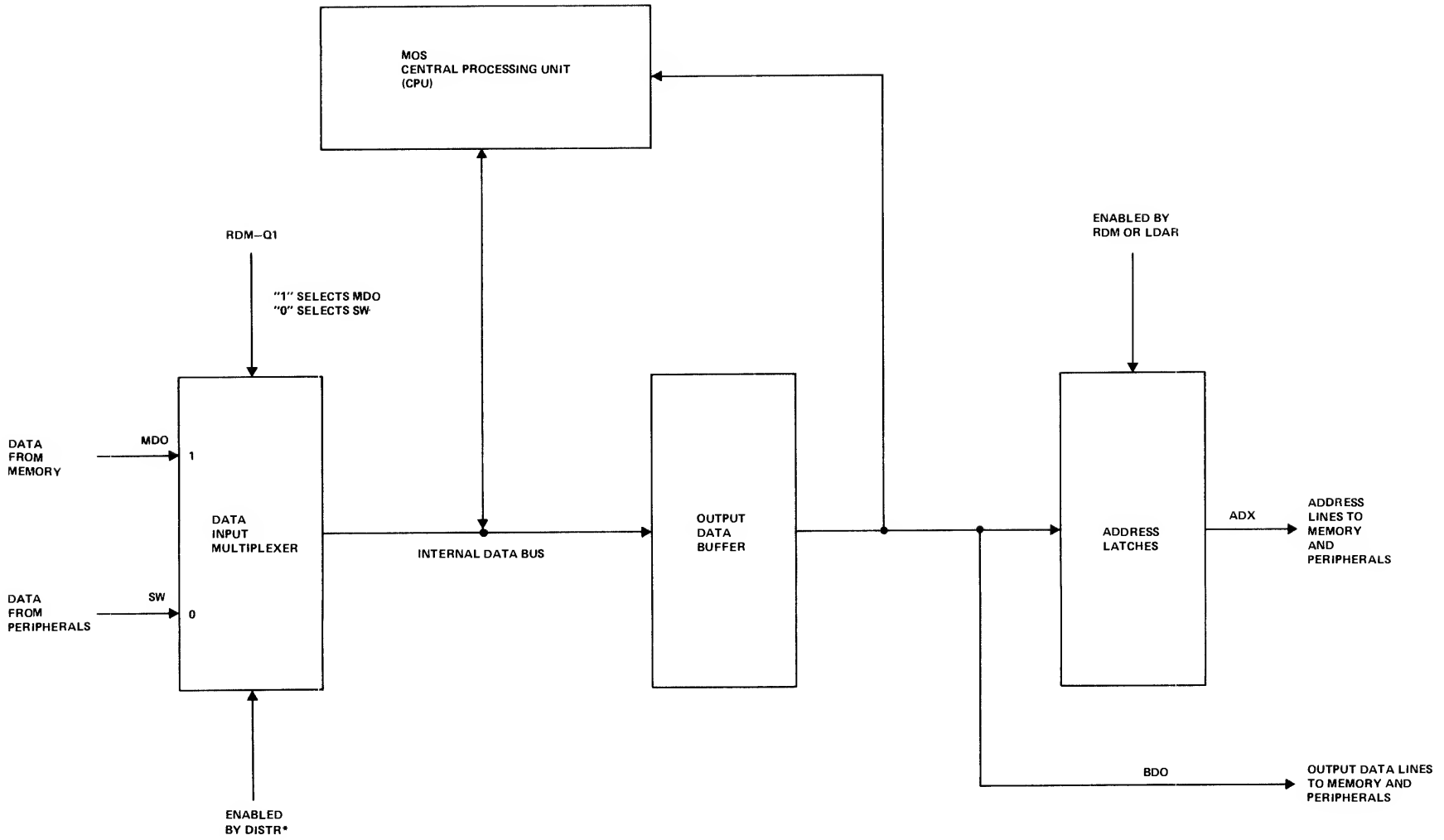


Figure 1-1. IMP-16C Bus Interface Structure

CHAPTER 1

GENERAL INFORMATION

This chapter discusses IMP-16C interface considerations, advisability of having a control panel for use with the IMP-16C, and interface methods and programs. More-detailed information on these subjects is presented in the remaining chapters.

1.1 IMP-16C INTERFACE CONSIDERATIONS

Peripheral devices communicate with the IMP-16C processor via the SW input bus and the BDO data out bus. A number of user control flags are available to control various I/O operations. In this application note, emphasis is placed on program-controlled operations to illustrate the simplicity of the interface requirements.

When designing a peripheral device interface, a wide range of possibilities exist between a single-purpose (economical) interface and a multi-purpose (expensive) interface. The single-purpose interface must rely on processor control to generate the necessary timing sequences, while the multi-purpose interface requires only a minimal amount of processor control. The trade-off between expense and hardware volume can only be determined by the user's needs. There is nothing inherent in the design of the IMP-16C bus system or the processor that requires a complex interface.

The teletype interface using control flags (described in chapter 3 of this note) is an example of a relatively simple peripheral interface that is flexible enough for all operations but requires a large degree of program control. This is acceptable for most applications where the processor is not doing other functions during teletype operations.

Figure 1-1 is a simplified block diagram of the I/O bus structure of the IMP-16C microprocessor. The timing of the various signals and data lines is given in figure 1-2. Data bits presented on the SW bus are accepted during T7 as determined by the DISTR* pulse. Output data is valid during T4. Input jump conditions and output control flags are valid at the start of T2. Control flags are reset at the beginning of T6. Addresses on the ADX bus start coming up during T3 and are valid at the start of T4.

When interfacing external devices with the IMP-16C, peripheral device addresses may be distinguished from memory addresses by the presence of the RDP or WRP flags that are pulsed during RIN and ROUT instructions, respectively.

Timing for RIN and ROUT instructions is shown in figure 1-3. Each of these instructions takes seven microcycles to execute. During the first microcycle (FETCH), the RDM flag enables the establishment of a memory address on the ADX lines to permit the fetching of the instruction. The next six microcycles effect the execution of the

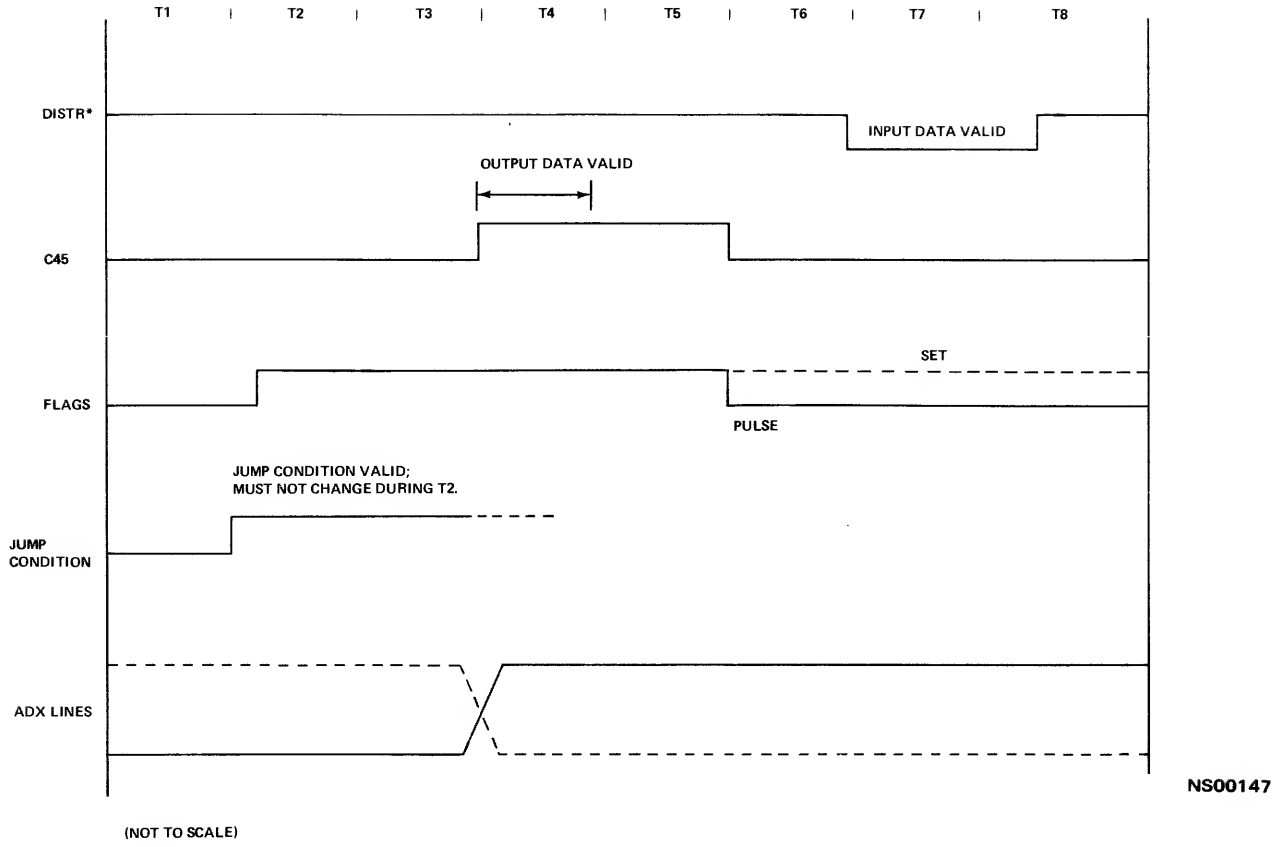


Figure 1-2. IMP-16C Timing Chart

instruction. In the sixth microcycle, the LDAR flag is pulsed, thus enabling the latching of the peripheral address on the ADX lines. These latched addresses are valid through the next microcycle, at which time the RDP or WRP flag is pulsed and the actual data transfer takes place.

During a ROUT instruction, the data present on the BDO lines at T4 of the seventh microcycle is the output data from AC0. During a RIN instruction, the data present on the SW lines at T7 of the seventh microcycle is the input data accepted by the processor and loaded into AC0.

1.2 IMP-16C CONTROL PANEL

The IMP-16C is self-contained for control operations; however, for some general-purpose applications a control panel may be needed or convenient to have for debugging programs, entering or retrieving data from the IMP-16C, verifying system operation, troubleshooting, and other operator-controlled purposes.

The control panel described here is one that is relatively uncomplicated and has a minimum number of components. A software service routine, resident in main memory, scans the panel switches and implements appropriate service actions according to switch positions. This control panel and its operations are described in chapter 2.

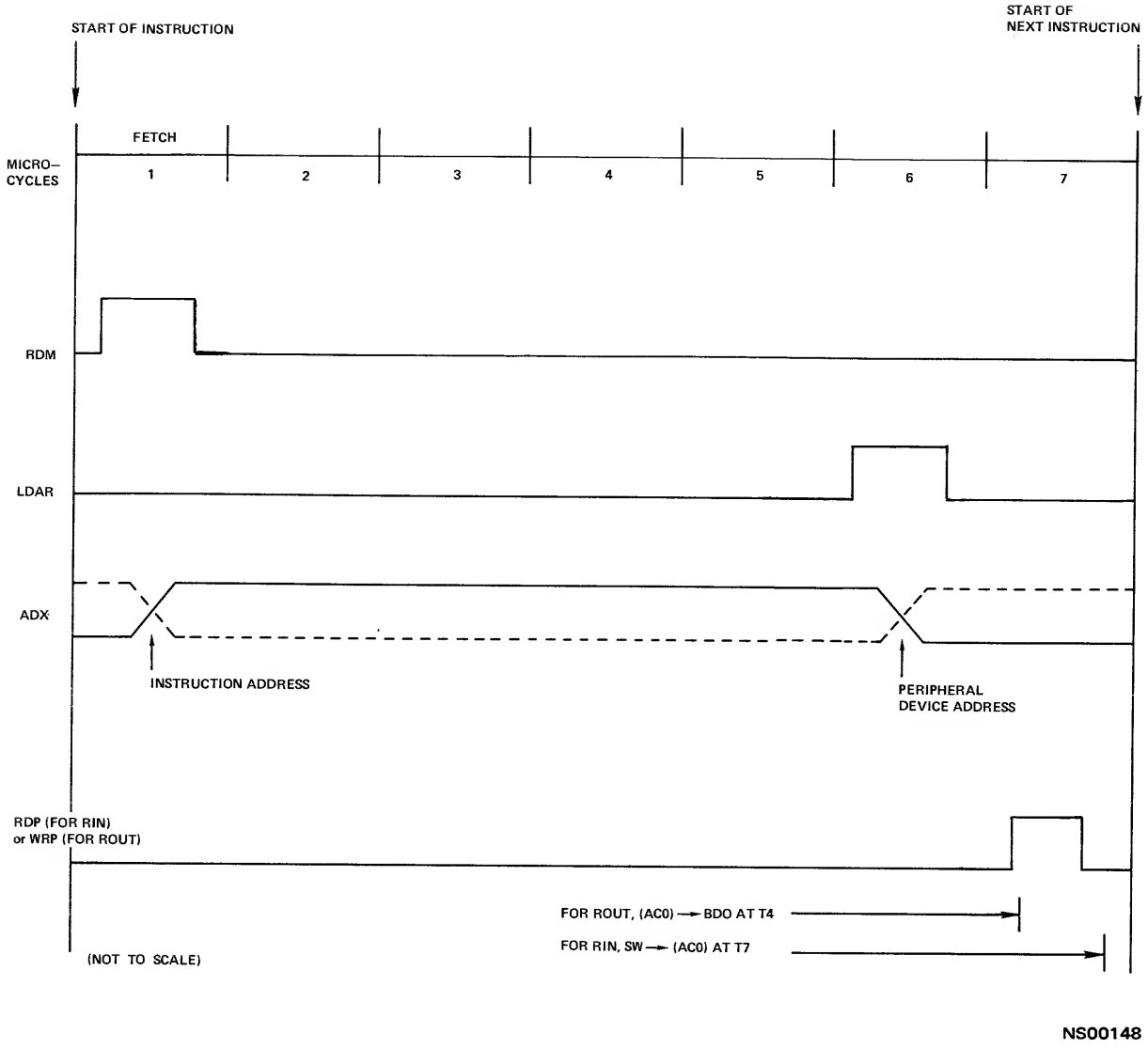


Figure 1-3. Timing Sequence for RIN and ROUT Instructions

1.3 INTERFACE METHODS AND PROGRAMS

Chapters 3 through 7 of this note present several interface methods and related programming considerations as well as a description of interrupt handling. Included are various sample programs that facilitate generating and loading program tapes.

A complete firmware package is available as a PROM and may be run on the IMP-16C (interfaced to a control panel). The listing of this package is given in appendix A.

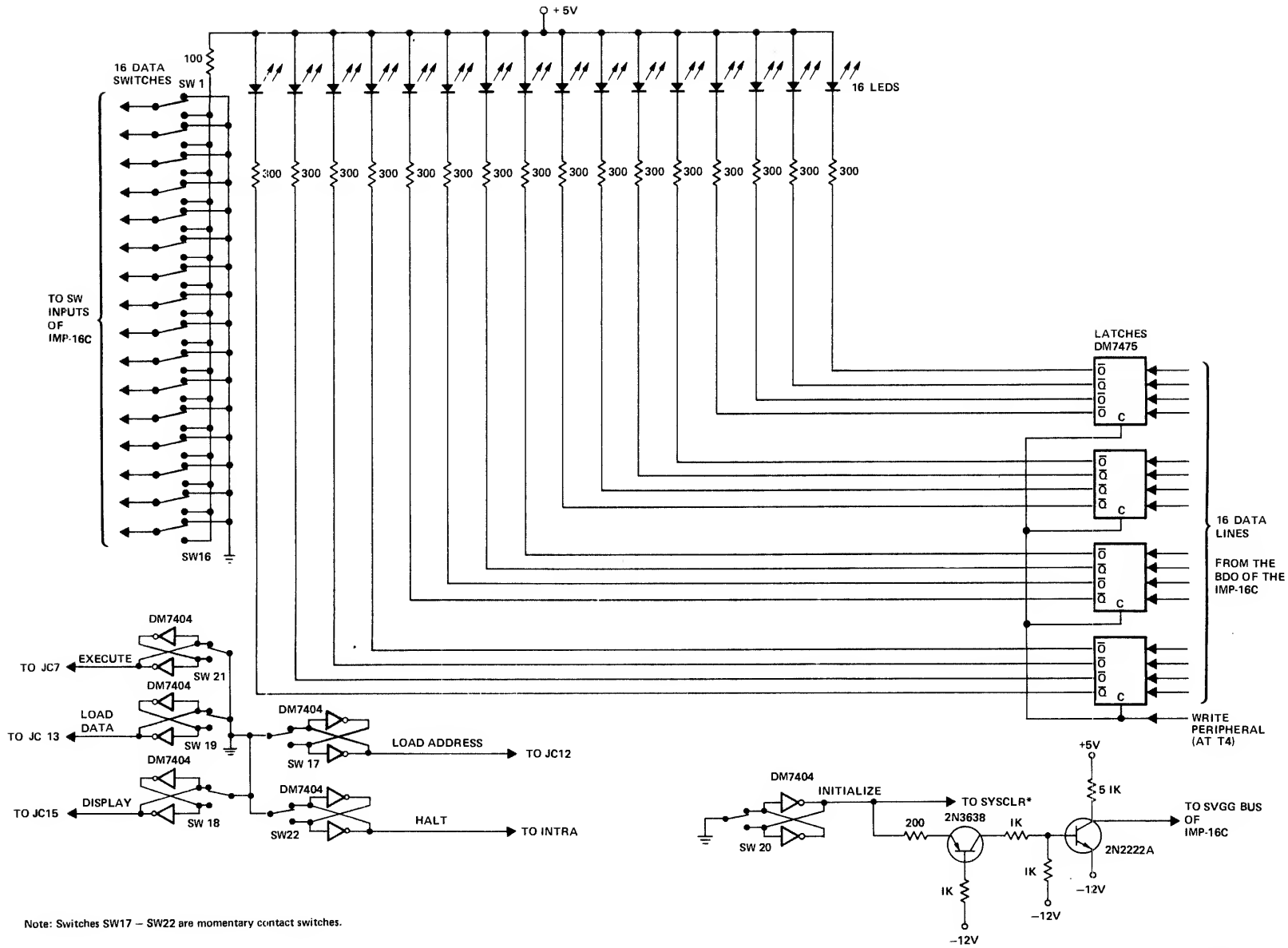


Figure 2-1. Control Panel Simplified Schematic Diagram

CHAPTER 2

CONTROL PANEL OPERATIONS

2.1 GENERAL INFORMATION

The control panel described here is a very simple means of entering and retrieving information from the IMP-16C card. It does not decode any device addresses; it works off the RDP and WRP flags pulsed by the RIN and ROUT instructions. The schematic diagram in figure 2-1 depicts the arrangement of the panel logic. With this panel, six momentary contact switches may be serviced by a software routine. The routine provided with this control panel assumes the following assignments.

	SWITCH	CONNECTED TO	FUNCTION
	SW21	JUMP CONDITION 7	EXECUTE(START)
	SW19	JUMP CONDITION 13	LOAD DATA
	SW18	JUMP CONDITION 15	DISPLAY
	SW17	JUMP CONDITION 12	LOAD ADDRESS
	SW22	INTRA	HALT (interrupt serviced)
Provides overriding initialization of CPU	SW20	SYSCLR*	INITIALIZE

The physical location of the various switches on the control panel board is shown in figure 2-2; the next few paragraphs describe operating procedures for this panel when it is used with the service program.

Figure 2-1 is a simplified version of the actual control panel logic schematic diagram (provided as part of the control panel kit IMP-16C/882, formerly CTLPLKIT). The switch numbers in the table above refer to switch positions called out on the original diagram.

2.2 OPERATING PROCEDURES

The following paragraphs provide the procedures for performing the various operations at the control panel.

2.2.1 Loading Main Memory

1. Set the Data Switches to the address of the desired starting location. (Note: locations 0 through 6 in memory are reserved for control program usage.)

2. Press LOAD ADDRESS. The selected address will be displayed on the Bit Display Lights.

3. Set the Data Switches to the data value to be loaded.

4. Press LOAD DATA. The value will be displayed, and the memory-location address will be incremented automatically.

5. Subsequent memory locations may be loaded in consecutive order by setting the desired bit pattern of the value to be loaded and pressing LOAD DATA after each setting.

2.2.2 Altering Memory Locations

1. Set the address of the location whose contents are to be altered on the Data Switches, and press LOAD ADDRESS.

2. Set the new value of the data on the Data Switches, and press LOAD DATA.

2.2.3 Examining Memory Locations

1. Set the address of the location whose contents are to be displayed on the Data Switches, and press LOAD ADDRESS. The selected address will be displayed on the Bit Display Lights.

2. Press DISPLAY. The contents of the selected location will be displayed, and the memory-location address will be incremented automatically.

3. Subsequent memory locations may be displayed in consecutive order by repeatedly pressing DISPLAY.

2.2.4 Loading The Accumulators

1. Set the Data Switches to the number of the accumulator to be loaded (X'0000 for AC0, X'0001 for AC1, and so on), and press LOAD ADDRESS.

2. Set the Data Switches to the value to be loaded, and press LOAD DATA.

3. Repeat steps 1 and 2 for each accumulator to be loaded.

2.2.5 Displaying Accumulator Values

1. Set the Data Switches to the number of the accumulator whose contents are to be displayed (similar to step 1 above), and press LOAD ADDRESS.

2. Press DISPLAY.

3. Repeat steps 1 and 2 for each accumulator whose contents are to be displayed.

2.2.6 HALT and Continue Operations

The panel HALT switch interrupts the processor and sets it in a HALT mode. If the Data Switches are set to X'0000 and EXECUTE is pressed, then the processor returns control to the control panel routine. Otherwise, an EXECUTE command causes the program to resume where it left off at the time of the HALT.

At the time of a HALT, the return address of the interrupted program is displayed on the Bit Display Lights. This can be noted and used as a resumption address if the normal return from interrupt is not used.

2.2.7 Executing A Program

1. Select the starting address of the program on the Data Switches.

2. Press EXECUTE.

After a program has been run, its results may be observed (looking at register values and examining specific memory locations) by coding a JMP X'00 instruction at the last executable instruction; that is, after entering a program, code X'2000 as the final instruction. This returns control to the control panel routine without altering the status of the registers. If register values need not be saved, the same effect can be achieved by pressing INIT.

2.2.8 Initialization

The function of the INIT switch is to clear the CPU and return control to the starting sequence. Main memory is not affected by this operation because it has separate power lines. Therefore, a program once loaded into the read/write memory remains unaltered until it is overwritten or power to the entire unit is shut off.

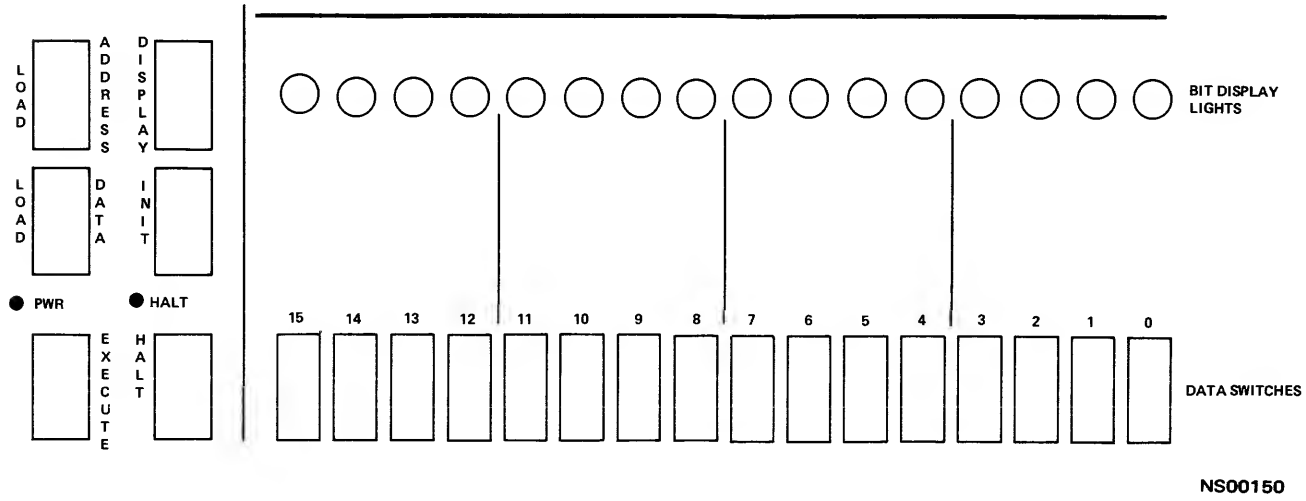


Figure 2-2. Control Panel Arrangement

2.3 CONTROL PANEL SERVICE ROUTINE

The program that effects the operations detailed in 2.2.1 through 2.2.7 works on a continuous scan of the front panel active switches. When the processor is powered up, it begins to execute instructions at memory location $FFFE_{16}$ (located in top page in a PROM/ROM). These instructions direct the processor into a 5-cycle wait loop that awaits a front panel command.

The front panel HALT function interrupts normal operation of the processor and sends control to an interrupt service routine. Program listings for the control panel service routine and the interrupt service routine are given on the following pages. It must be noted that this simple control panel by itself is not intended to be a software debug aid, but rather it is a means of manual entry of information into the IMP-16C. As such, when using any of the routines described in this application note, the directions supplied must be followed closely.

NOTE

All numbers that represent data are written in hexadecimal format. The notation for this format is either the subscript 16 following the number or the prefix X' preceding the number. Example: $4A08_{16}$ means the same as X'4A08.

```
FF86                    .PAGE       'CONTROL PANEL SERVICE ROUTINE'
FF86              ;
FF86              ;                 VERSION 2, APRIL 25, 1973
FF86 FFAD A             .=X'FFAD
FFAD 21B4  A JSTRT:     .WORD X'21B4
FFAE 21ED  A JINTR:     .WORD X'21ED
FFAF 0005  A FIVE:      .WORD  5
FFB0 0001  A ONE:       .WORD  1
FFB1 8DFC  A BEGIN:     LD 3, JINTR
FFB2 AC01  A            ST 3,X'01;      LOAD LOCATION 1 WITH JUMP TO INTERRUPT
FFB3 8DF9  A            LD      3,JSTRT
FFB4 AC00  A            ST 3,X'00;      LOAD LOCATION 0 WITH JUMP TO CONTROL PANEL
FFB5 2929  A START:     JSR SAVE        ; SAVE ACCUMULATORS.
FFB6 0900  A SET:       SFLG 1          ; ENABLE INTERRUPT SYSTEM.
FFB7 0600  A ROUT:      ROUT 0
FFB8 1C04  A WAIT:      BOC C12,LA      ;  'LOAD ADDRESS' SWITCH.
FFB9 1D0A  A            BOC C13,LD      ;  'LOAD DATA' SWITCH.
FFBA 1710  A            BOC C7, EX      ;  'EXECUTE' SWITCH.
FFBB 1F1F  A            BOC C15, DISP   ;  'DISPLAY' SWITCH.
FFBC 21FB  A            JMP .-4         ;  RETURN TO WAIT LOOP.
FFBD 1CFF  A LA:        BOC C12,LA      ;  CHECK RELEASE.
FFBE 0400  A            RIN 0           ;  READ SWR.
FFBF 3281  A            RCPY 0,2        ;  SAVE ADDRESS IN AC2.
FFC0 3381  A            RCPY 0,3
FFC1 E1ED  A            SKG 0,FIVE;     PREVENTS LOADING OF RESERVED LOCATIONS
FFC2 120D  A            BOC C2, RSRVE
FFC3 21F3  A            JMP ROUT
FFC4 1DFF  A LD:        BOC C13,LD;     CHECK RELEASE FOR LOAD DATA SWITCH
FFC5 0400  A            RIN 0;          READ SWITCHES
FFC6 F938  A            SKNE 2,LAST6;     PREVENTS LOADING LOCATION 6.
FFC7 21F0  A            JMP WAIT
FFC8 A200  A            ST 0,(2)        ;  LOAD MEMORY
```

```
FFC9 C9E6 A           ADD 2, ONE;   INCREMENT ADDRESS
FFCA 21EC A           JMP ROUT
FFCB 17FF A EX:       BOC C7, EX    ;  CHECK RELEASE
FFCC 0400 A           RIN 0
FFCD 4000 A           PUSH 0        ;  SAVE JUMP ADDRESS IN STACK.
FFCE 2918 A           JSR RSTOR
FFCF 0200 A           RTS 0         ;  FAKING AN INDIRECT JUMP.
FFD0 0600 A RSRVE:    ROUT 0
FFD1 1D05 A           BOC C13, LDAC
FFD2 1F01 A           BOC C15,DISPAC
FFD3 21FD A           JMP .-2
FFD4 1FFF A DISPAC:   BOC C15,DISPAC; DISPLAY ACCUMULATOR ROUTINE
FFD5 8302 A           LD 0,2(3)
FFD6 21E0 A           JMP ROUT
FFD7 1DFF A LDAC:     BOC C13,LDAC
FFD8 0400 A           RIN 0
FFD9 A202 A           ST 0,2(2)
FFDA 21DC A           JMP ROUT
FFDB 1FFF A DISP:     BOC C15,DISP
FFDC 8300 A           LD 0,(3)
FFDD CDD2 A           ADD 3, ONE;   INCREMENT ADDRESS
FFDE 21D8 A           JMP ROUT
FFDF          ;
FFDF A002 A SAVE:     ST 0,X'02     ;  SAVE AC0 - AC3 IN LOCATIONS X'02 - X'05.
FFE0 A403 A           ST 1,X'03
FFE1 A804 A           ST 2,X'04
FFE2 AC05 A           ST 3,X'05
FFE3 0080 A           PUSHF
FFE4 4500 A           PULL 1
FFE5 A406 A           ST 1,X'06;    SAVE FLAGS IN LOCATION 6.
FFE6 0200 A           RTS 0
FFE7          ;
FFE7 8406 A RSTOR:    LD 1,X'06
FFE8 4100 A           PUSH 1
FFE9 0280 A           PULLF
FFEA 8002 A           LD 0,X'02;    RESTORE ACCUMULATORS.
FFEB 8403 A           LD 1,X'03
FFEC 8804 A           LD 2,X'04
FFED 8C05 A           LD 3,X'05
FFEE 0200 A           RTS 0
FFEF                  .PAGE         'INTERRUPT SERVICE ROUTINE FOR HALT AND STACKFULL'
FFEF          ;
FFEF 180B A INTR:     BOC C8,STFL
FFF0 29EE A           JSR SAVE;     SAVE ACCUMULATORS IN LOCATIONS 2,3,4 AND 5.
FFF1 4400 A           PULL 0
FFF2 4000 A           PUSH 0
FFF3 0600 A           ROUT 0;       INTERRUPT ADDRESS IS DISPLAYED.
FFF4 0000 A           HALT
FFF5 0400 A           RIN 0;        READ SWITCHES
FFF6 1102 A           BOC C1,.+3;   IF SWITCHES ARE SET TO ALL ZEROS, 'EXECUTE' CAUSES
FFF7 29EF A           JSR RSTOR;    A RETURN TO THE CONTROL PANEL ROUTINE.
FFF8 0100 A           RTI ;         IF SWITCHES ARE SET TO ANY NON-ZERO NUMBER, 'EXECUTE
FFF9          ;                     CAUSES A NORMAL RETURN FROM INTERRUPT.
FFF9 4400 A           PULL 0
FFFA 21BB A           JMP SET;      RETURN TO CONTROL PANEL ROUTINE.
FFFB 4CFF A STFL :    LT 0,-1
FFFC 0600 A           ROUT 0        ;  NON RECOVERABLE SITUATION; "INITIALIZE" RETURNS
FFFD 0000 A           HALT          ;  CONTROL TO PANEL.
FFFE 21B2 A           JMP BEGIN
FFFF 0006 A LAST 6:   .WORD 6
 000                  .END
```

2.4 CONTROL PANEL WIRE LIST

The wire list for the connections between the control panel card and the IMP-16C is provided in table 2-1.

Note that the switches from the control panel are brought out directly to terminals. When these lines are connected to the IMP-16C, no other lines may be connected to the same bus. Therefore, if it is desired to hook up other equipment on the input data bus, the switch lines from the control panel must be passed through buffers with tri-state outputs that may be disabled.

Table 2-1. Wire List

SIGNAL	FROM	TO	SIGNAL	FROM	TO	SIGNAL	FROM	TO
	Control Panel	IMP-16C		Control Panel	IMP-16C		Control Panel	IMP-16C
Ground	E1	1-4, 141-144	BDO 05	E17	56	BDO 13	E33	74
+5v	E2	5-8, 137-140	BDO 06	E18	65	BDO 15	E34	70
SW 00	E3	79	BDO 07	E19	63	BDO 14	E35	69
SW 01	E4	84	SW 08	E20	90	WRPB	E36	53
BDO 00	E5	60	SW 09	E21	92	EXEC	E37	123
BDO 01	E6	58	SW 11	E22	93	INTRA (1)	E38	15
SW 02	E7	81	SW 10	E23	91	LDM (2)	E39	119
BDO 02	E8	64	BDO 09	E24	67	HLT* (3)	E40	125
BDO 03	E9	61	BDO 08	E25	27	LDA (4)	E41	129
SW 03	E10	86	BDO 10	E26	73	SYSCLR (5)	E42	-
SW 04	E11	98	BDO 11	E27	68	SYSCLR*	E43	126
SW 05	E12	103	SW 12	E28	95	C 45 (6)	E44	-99
SW 07	E13	105	SW 13	E29	96	DISP (7)	E45	107
SW 06	E14	100	SW 15	E30	106	-12v	E46	31, 32
WRPA	E15	66	SW 14	E31	97	GROUND	E47	1-4, 141-144
BDO 04	E16	59	BDO 12	E32	75	-12v SWITCHED	E48	11, 12

NOTES:
(1) INTERRUPT; WIRED TO CONTROL PANEL HALT
(2) WIRED TO JC13 ON IMP-16C
(3) HALT INDICATOR FLAG
(4) WIRED TO JC12 ON IMP-16C
(5) UNUSED
(6) C45 TIMING SIGNAL
(7) WIRED TO JC15 ON IMP-16C

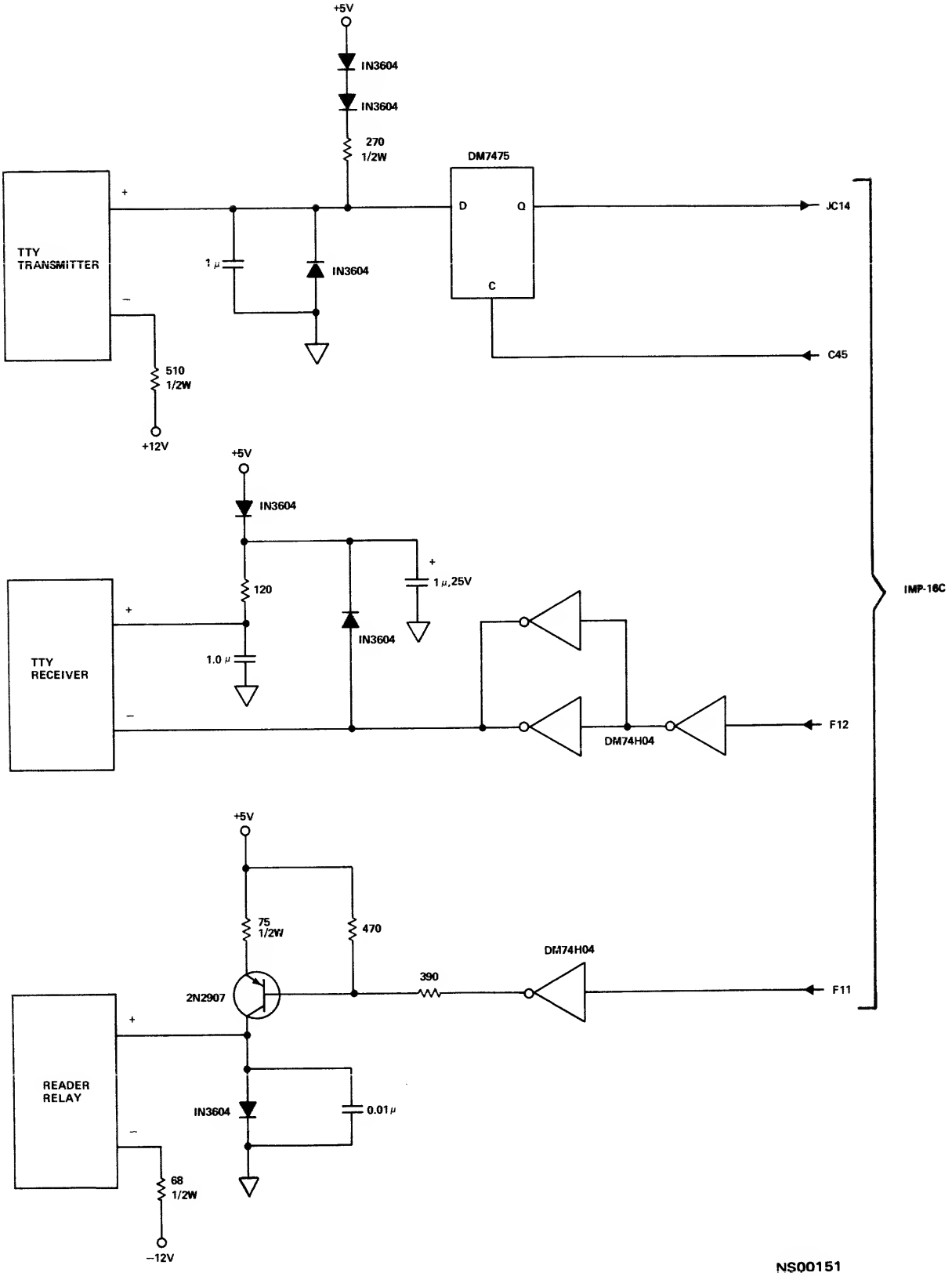


Figure 3-1. Serial Teletype Interface, Flag Controlled

CHAPTER 3

SERIAL TELETYPE INTERFACES

3.1 GENERAL INFORMATION

Two program-controlled teletype interfaces are presented in this chapter. The first is a simple, economical interface that employs control flags. The second interface requires more hardware since it employs device address decoding, but this interface has the advantage of being completely software-supported and does not tie up user flags. Each of these, along with sample programs, is described in the following paragraphs.

3.2 PROGRAM-CONTROLLED INTERFACE USING FLAGS

A very simple teletype interface can be built using two thirds of a DM7404 package and a few discrete elements. This interface permits communication with the IMP-16C via a user jump condition and two user control flags. This approach eliminates the need for device address decoding and, consequently, turns out to be a very economical implementation. Since the address fields of the RIN and ROUT instructions are not used, AC3 is not tied up any more and can be used freely by the programmer. The circuit schematic is given in figure 3-1.

The listings provided under paragraphs 3.2.1 and 3.2.2 describe a character-read routine (RECV) and a character-transmit routine (SEND) that receives/sends bit-serial information between the TTY and the IMP-16C. User jump condition 14 (JC14) is used for data-in and flag 12 (F12) is used for data-out. An additional flag (F11) is used as a reader-enable control signal.

NOTE

1. In figure 3-1, an optional 7475 is used to synchronize the jump condition input into the IMP-16C to assure that the signal is stable between successive T4 times of any microcycle sequence. This is recommended to prevent the rare (but possible) occurrence of the jump condition input making a transition during the leading edge of phase 2. On the IMP-16C/200, C/300 this circuit is already provided.

2. The mnemonic TTY is used frequently to denote "teletype."

3.2.1 Teletype Transmit Character Routine

This routine takes one character (right justified in AC0) and sends it to the TTY. Since this is written as a subroutine, accumulators 0, 1, and 2 are saved on the stack before being used in the routine as temporary storage areas.

```
        .PAGE   'TRANSMIT CHARACTER ROUTINE'
        XMIT = 4                ; TELETYPE TRANSMIT FLAG
SEND:   PUSH    2               ; SAVE ACCUMULATORS.
        PUSH    1
        SFLG    XMIT            ; SEND START BIT.
        JSR     DELAY           ; DELAY INTO FIRST DATA BIT.
        LI      2,8             ; SET BIT COUNT.
PUT:    PFLG    XMIT            ; CLEAR TRANSMIT FLAG.
        BOC     3,$XX           ;
        SFLG    XMIT            ; SEND DATA BIT.
$XX:    JSR     DELAY
        SHR     0,1
        AISZ    2,-1            ; TEST TO SEE IF DONE.
        JMP     PUT
        PFLG    XMIT            ; SEND TWO STOP BITS.
        JSR     DELAY
        JSR     DELAY
        PULL    1
        PULL    2
        RTS
```

The DELAY subroutine below is used in the routine above and in the RECV routine to provide the required delay between the teletype bits that are being processed serially.

```
DELAY:  LD      1,V2            ; LOAD TIMING PARAMETER
        AISZ    1,-1
        JMP     .-1
        RTS
V1:     .WORD   01B1
V2:     .WORD   035E
```

3.2.2 Teletype Receive Character Routine

This routine takes one character from the TTY and loads it into AC0 (right justified).

A loader that reads IMP-16 Assembler-generated load modules (RLMs) and that loads memory is described in chapter 5. The programs described in section 3.2 may be used in conjunction with the control panel described in chapter 2.

```
        .PAGE   'TELETYPE GET CHARACTER ROUTINE'
        JC14  = 14              ; INPUT JUMP CONDITION.
        READR =  3              ; READER ENABLE FLAG.
        C1    =  1
        C2    =  2
RECV:   PUSH    1               ; SAVE ACCUMULATORS.
        PUSH    2
        PFLG    2               ; DISABLE LINK.
        LI      2,8             ; SET COUNT FOR 8 BITS.
        SFLG    READR
        BOC     JC14,.+2        ; TEST FOR START BIT.
        JMP     .-1             ; LOOP UNTIL FOUND.
        LD      1,V1            ; LOAD TIMING PARAMETER.
        JSR     DELAY+1         ; DELAY HALF BIT TIME.
        PFLG    READR
        BOC     JC14,.+2        ; TEST FOR DATA BIT.
        JMP     RECV+2
REP:    JSR     DELAY
        SHR     0,1
        BOC     JC14,.+2
        OR      0,H8000
        AISZ    2,-1            ; DECREMENT COUNT.
        JMP     REP
        JSR     DELAY
        SHR     0,8
        PULL    2
        PULL    1
        RTS

H8000:  .WORD   08000           ; MASK WORD
```

3.3 PROGRAM-CONTROLLED INTERFACE USING DEVICE ADDRESSES

A more-conventional (in terms of minicomputer type of usage) TTY interface uses a peripheral device address decoder. This approach obviously requires more hardware, but it has the advantage of being software-supported because IMP-16L TTY routines may be used directly and does not tie up any user flags.

This teletype interface is a full-duplex, bit-serial communication path that allows the processor to send or receive serial bit streams to and from the teletype. The formatting and timing of the bit stream must be controlled by the processor. All communication between the processor and the interface is over the system data bus.

Figure 3-3 shows the hardware required for this type of interface. The driver circuits are the same as described in 3.2. The 8-bit device address and 3-bit order field are decoded by DM7430 gate and a DM74155 3-to-8-line decoder.

The data received by the teletype may be read over the system data bus in bit position 15 by use of the RIN instruction. The data format is shown in figure 3-2. Bits 0 through 14 are undefined and should be masked by the program. The bits comprising the character will be input serially, least significant bit first.

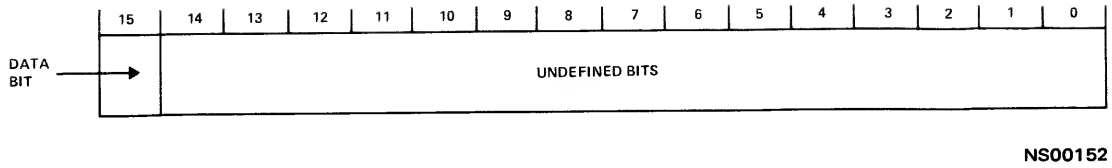


Figure 3-2. Teletype Data Word Format

The teletype transmit circuits may be instructed to transmit a 1 or a 0 by use of the ROUT instruction. Once set to a value, the circuits continue to transmit that value until instructed to transmit another value or cleared with a RESET order. The transmit value is set from bit 15 of the data bus. Bits 0 through 14 are not used and may be any value. The bits comprising the character should be transmitted serially, least significant bit first.

The paper tape reader control may be used for teletypes that have a paper tape reader control circuit. This optional feature allows program control of the tape reader; this is especially desirable for applications where the data are processed as they are read as the reader may be stopped during data processing. Six order codes are acknowledged by the teletype interface. These are listed in table 3-1. The effect of each order is explained below. Those orders marked with an asterisk are not used in the program examples given here.

RIN Code 2 - Bit 15 of the data bus is set equal to the output of the teletype. The processor, in turn, will transfer the data bus to AC0. A binary 1 represents a teletype mark, and a binary 0 represents a teletype space. Bits 0 through 14 are undefined.

RIN Code 4 - The paper tape reader is turned on.

RIN Code 5 - The interrupt request and interrupt enable flags are turned off. The teletype output is set to the idle (marking) state. The paper tape reader enable is turned off.

*RIN Code 6 - The teletype responds to the "Interrupt Select Status 1" order by setting data bit 7 equal to the state of the interrupt request flag.

*ROUT Code 1 - The interrupt enable flag is turned on.

ROUT Code 3 - The teletype transmit circuit is set to the value of data bit 15.

ROUT Code 4 - The paper tape reader is turned on.

ROUT Code 5 - The interrupt request and interrupt enable flags are turned off. The teletype output is set to idle (marking) state. The paper tape reader enable is turned off.

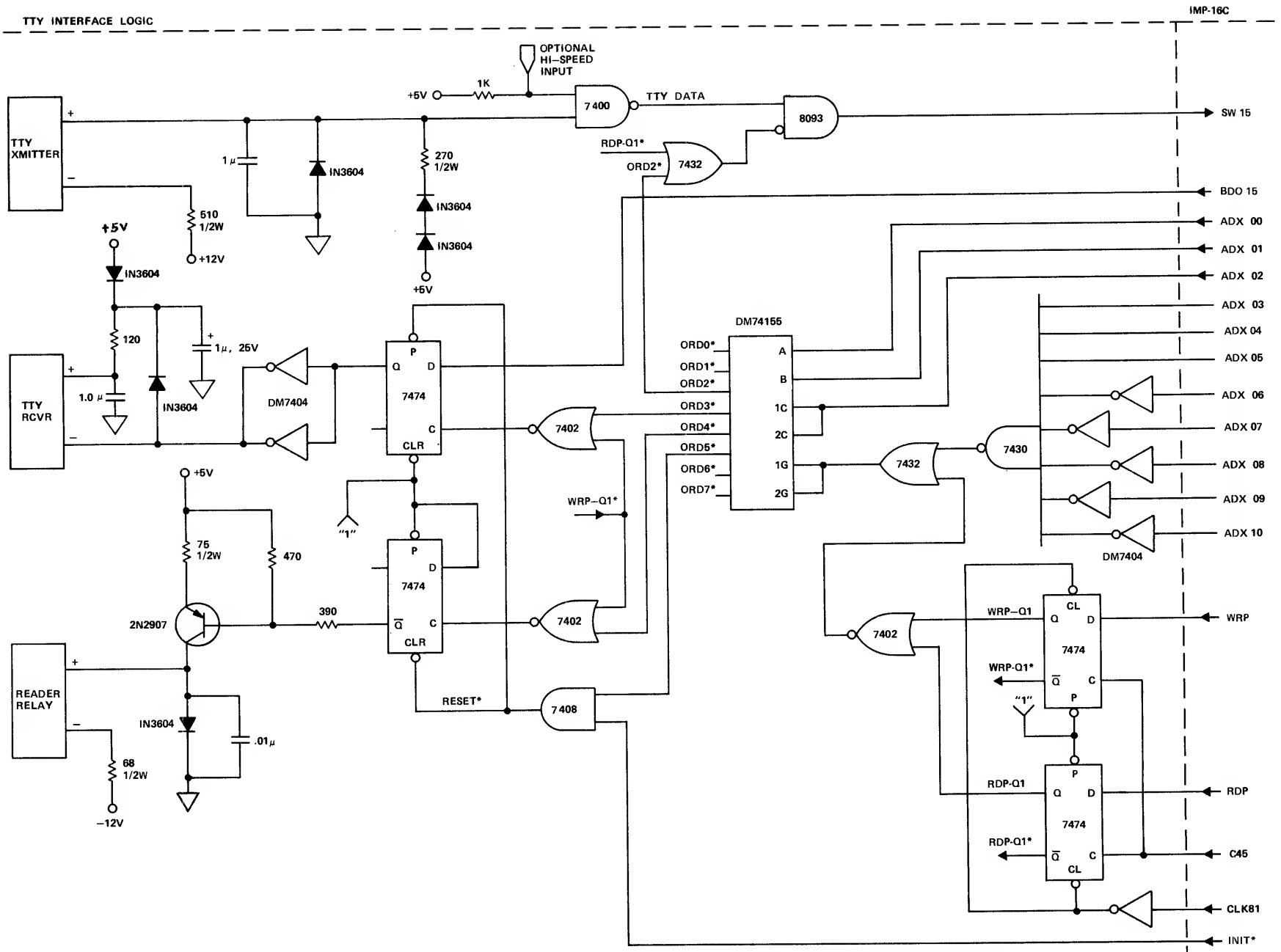


Figure 3-3. Serial Teletype Interface, Device Address-Controlled

NS00153

Table 3-1. Serial Teletype Order Codes

Instruction	Order Code	Action
RIN	000	NO ACTION
	001	NOT ALLOWED
	010	READ BIT FROM TTY
	011	NOT ALLOWED
	100	START TAPE READER
	101	RESET
	110	INT STATUS TO BIT 7*
	111	NO ACTION
ROUT	000	NO ACTION
	001	SET INTEN = DATA BIT 15*
	010	NOT ALLOWED
	011	WRITE BIT TO TTY
	100	START TAPE READER
	101	RESET
	110	NOT ALLOWED
	111	NO ACTION

The program listings that follow show the corresponding receive and transmit routines for this hardware. For these programs, the teletype has been assigned a peripheral address of 0038_{16}, and the various order codes are given in table 3-1.

3.3.1 Teletype Transmit Character Routine

This routine takes one character (right justified in AC0) and sends it to the TTY. Since this is written as a subroutine, the contents of accumulators 1 and 2 are saved in memory before the accumulators are used in the routine as temporary storage areas.

```
        .PAGE   'TELETYPE PUT CHARACTER ROUTINE'
        TTYAD = 7*8             ; TELETYPE ADDRESS
        SEND  = 3
PUTC:   ST      1,7             ; SAVE ACCUMULATORS.
        ST      2,8             ;
        PFLG    2
        LD      1,V1
        JSR     DELAY+1
        LI      2,9
        LI      3,TTYAD
        ROUT    SEND
LP2:    JSR     DELAY
        AISZ    2,-1            ; CHECK TO SEE IF DONE
        JMP     .+2
        JMP     DONE
        ROR     0,1
        ROUT    SEND
        JMP     LP2
DONE:   LI      0,-1            ; SEND STOP BIT.
        ROUT    SEND
        JSR     DELAY
        LD      1,7
        LD      2,8
        RTS                     ; RESTORE ACCUMULATORS.
```

3.3.2 Teletype Receive Character Routine

This routine takes 8 bits of serial data from the teletype transmitter interface and packs them into AC0 with the bits right-justified.

```
        .PAGE   'TELETYPE GET CHARACTER ROUTINE'
        TTYAD = 7*8
        RESET = 5
        RDREN = 4
        READ  = 2
GETC:   ST      2,8             ; SAVE AC1 AND AC2 IN
        ST      1,7             ; LOCATIONS 7 AND 8.
        PFLG    2
        LI      3,TTYAD
        ROUT    RESET
        LI      2,8             ; SET BIT COUNT TO 8.
        ROUT    RDREN
        RIN     READ
        BOC     2,.+2           ; TEST FOR START BIT
        JMP     .-2
        LD      1,V1
        JSR     DELAY+1
        RIN     READ
        BOC     2,.+2
        JMP     GETC+3
```

(listing continued)

```
LP1:    JSR     DELAY2          ; DELAY ONE BIT TIME.
        RIN     READ
        AND     0,MASK
        SHR     1,1
        RXOR    0,1
        AISZ    2,-1;           ; TEST TO SEE IF DONE.
        JMP     LP1
        JSR     DELAY           ; DELAY INTO FIRST STOP BIT.
        SHR     1,8
        RCPY    1,0
        LD      1,7             ; RESTORE ACCUMULATORS.
        LD      2,8;
        RTS
MASK:   .WORD   X'8000
```

The GETC and PUTC routines use the same type of DELAY subroutine as the RECV and SEND routines of the previous section. Locations 7 and 8 of main memory are used as temporary storage locations.

```
DELAY:  LD      1,V2
        AISZ    1,-1
        JMP     .-1
        RTS
DELAY2: LD      0,V2
        AISZ    0,-1
        JMP     .-1
        RTS
V1:     .WORD   433             ; DELAY PARAMETERS
V2:     .WORD   862
```

3.3.3 Teletype Get Character and Echo Routine

This routine takes 8 bits of serial data from the teletype transmitter, packs them into AC0 with the bits right-justified, and also echoes the character back to the teletype receiver. The echo operation is done bit-by-bit so that the maximum rate of character processing can be achieved. The program listing follows on page 3-9.

3.4 TELETYPE TIMING PARAMETERS

All the teletype programs described in this chapter utilize software delay routines to time the serial transmission of teletype data bits. These routines have timing parameters that provide 1/2-bit and 1-bit delays based on a teletype speed of 110 bits/second. The following example shows how the parameter V1 in the delay routine is derived. A similar calculation yields V2.

Time for AISZ instruction = $4 \times 1.4 + 0.35\ \mu s = 5.95\ \mu s$

Time for JMP instruction = $3 \times 1.4 + 0.35\ \mu s = 4.55\ \mu s$

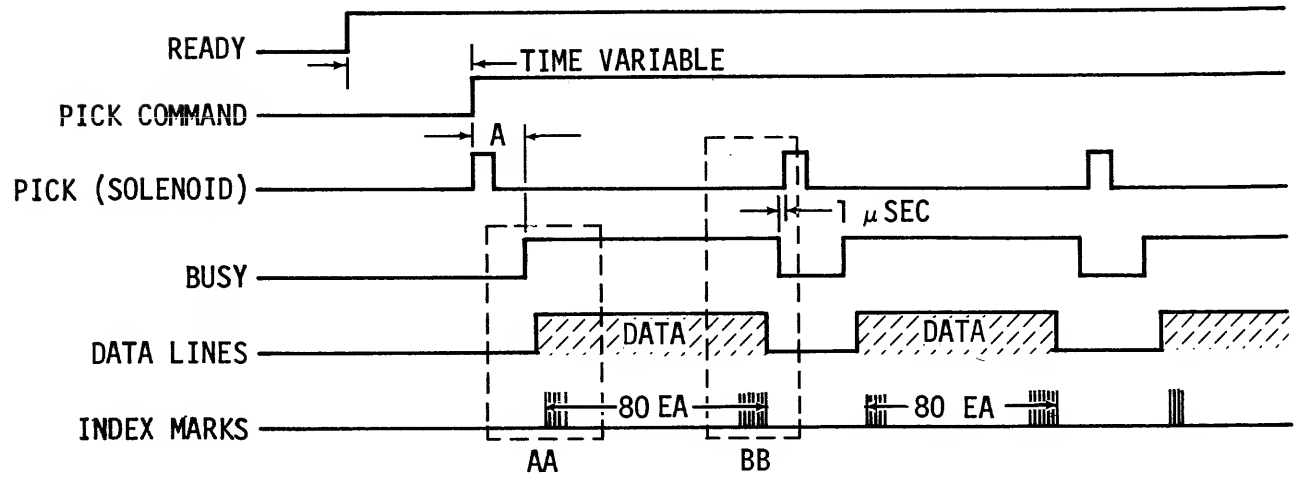
1/2-bit delay time = 4.545 ms

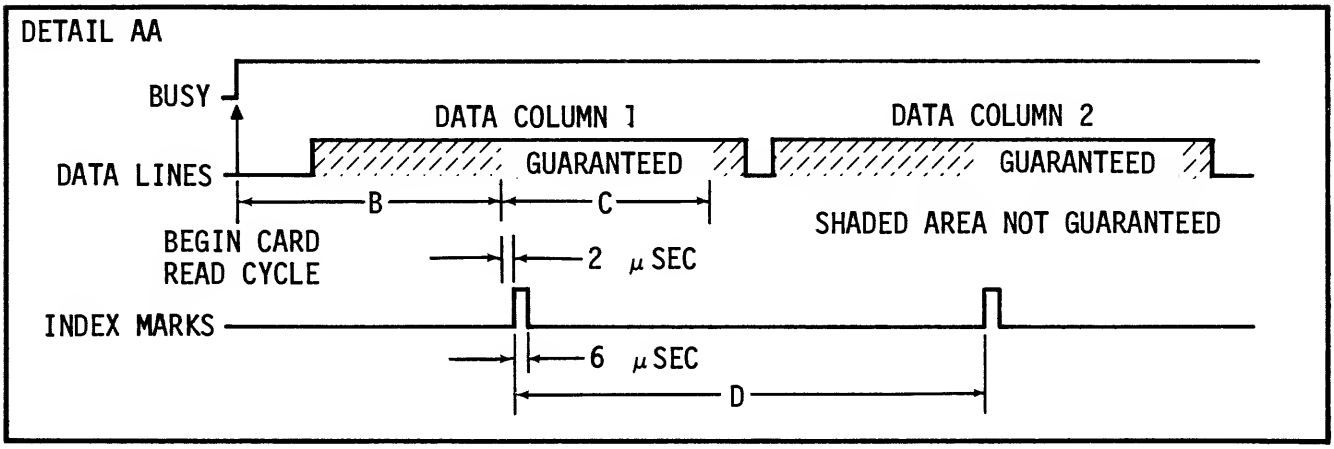
$$V1 = \frac{4.545 \times 10^{-3}}{(5.95 + 4.55)10^{-6}} = 433$$

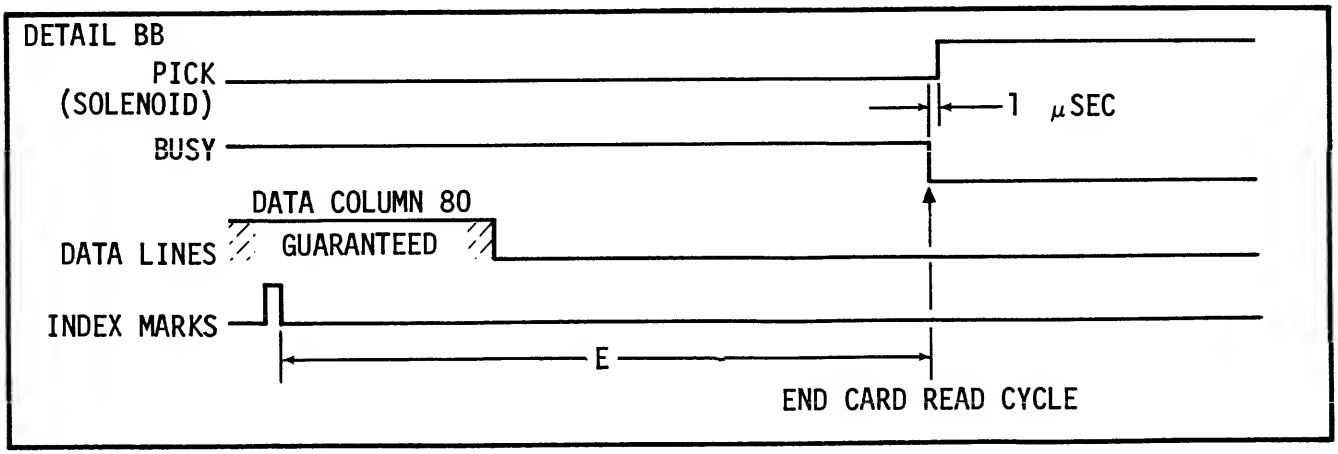
```
        .PAGE   'TELETYPE GET CHARACTER ROUTINE WITH ECHO'
;
TTYAD   =       7*8
READ    =       2
SEND    =       3
RDREN   =       4
RESET   =       5
;
GECHO:  ST      AC2,8           ; SAVE AC1 AND AC2 IN
        ST      AC1,7           ; LOCATIONS 7 AND 8
        PFLG    2
        LI      AC3,TTYAD
        ROUT    RESET           ; RESET TELETYPE
        LI      AC2,8           ; SET BIT COUNT TO 8
        ROUT    RDREN           ; ENABLE READER
        RIN     READ
        BOC     2,.+2           ; TEST FOR START BIT
        JMP     .-2
        LD      AC0,V1
        JSR     DELAY0+1        ; DELAY 1/2 BIT TIME
        RIN     READ            ; TEST IF START BIT IS STILL THERE
        BOC     2,.+2           ; BRANCH IF GOOD START BIT
        JMP     GECHO+3
LP3:    ROUT    SEND            ; ECHO BIT
        JSR     DELAY0          ; DELAY ONE BIT TIME
        RIN     READ
        AND     AC0,MASK        ; MASK UNWANTED BITS
        SHR     AC1,1           ; SHIFT DATA
        RXOR    AC0,AC1         ; ADD NEW BIT TO DATA
        AISZ    AC2,-1          ; TEST TO SEE IF DONE
        JMP     LP3
        ROUT    SEND            ; ECHO LAST BIT
        JSR     DELAY0          ; DELAY INTO FIRST STOP BIT
        LI      AC0,-1
        ROUT    SEND            ; SEND STOP BIT
        SHR     AC1,8           ; SHIFT DATA INTO RIGHT 8 BITS
        RCPY    AC1,AC0         ; PUT CHARACTER IN AC0
        LD      AC1,7           ; RESTORE ACCUMULATORS
        LD      AC2,8
        RTS


MASK:   .WORD   X'8000
V1:     .WORD   01B1
V2:     .WORD   035E
DELAY0: LD      AC0,V2          ; DELAY SUBROUTINE (AC0)
        AISZ    AC0,-1
        JMP     .-1
        RTS
```

TIMING CHART

MODEL	A (msec)	B (msec)	C (μsec)	D (μsec)	E (msec)
M200	24	6.25	1314	2014	8.05
M300	24	2.60	435	870	102.66
M600	24	2.60	435	870	3.48







NS00154

Figure 4-1. Standard Interface Timing for Documation M Card Readers

CHAPTER 4

CARD READER INTERFACE

4.1 GENERAL INFORMATION

Two simple program-controlled card reader interfaces are described in this chapter. The DOCUMATION 600 (or 300) card reader has been used in the examples; timing for this card reader is shown in figure 4-1. The IMP-16C processor initiates the operation by sending out a "pick" command to fetch a card. After the card has been picked, the card reader sends out 80 index marks that signify the start of each column of data. The IMP-16C program detects the presence of these marks and reads and stores each column of data into an 80-word buffer.

There are two ways to effect this operation: one method that uses only five IC packages makes use of two control flags and no device address decoding. This approach has the advantage of being economical and simple to implement, but it ties up two of the six available user control flags. The second method does not use any flags but requires more hardware to achieve the same purpose. Each of these two methods is described in the following paragraphs.

4.2 PROGRAM CONTROLLED INTERFACE USING CONTROL FLAGS

The following program segment shows how to read an 80-column card and store the data into a buffer. The hardware for this is shown in figure 4-3. For this method, no peripheral addresses are required; as a matter of fact, no RIN or ROUT instructions are required for anything other than actual input of data. All other control operations are effected with the SFLG and PFLG instructions.

The circuits shown in figure 4-3 are used with the control panel of chapter 2. When used in this mode, the switches from the control panel are wired into the SW inputs of the DM8123 multiplexers (as shown in figure 4-3) instead of directly to the SW lines of the IMP-16C card. Flag 14 (under program control in the RDCARD routine) selects either the switches or the card-reader for data input to the IMP-16C. The block diagram below shows the arrangement of this interface and a control panel with the IMP-16C.

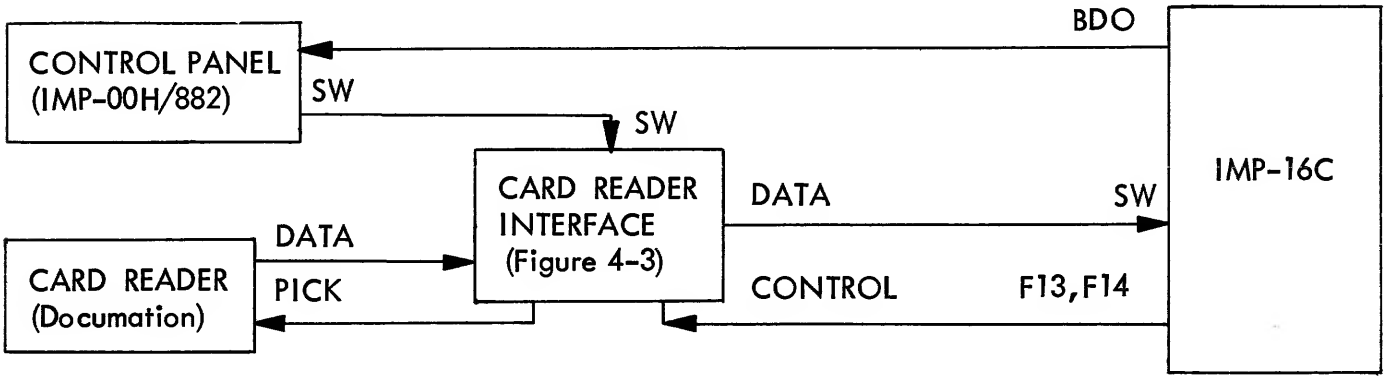


Figure 4-2. IMP-16C and Control Panel Interface, Block Diagram

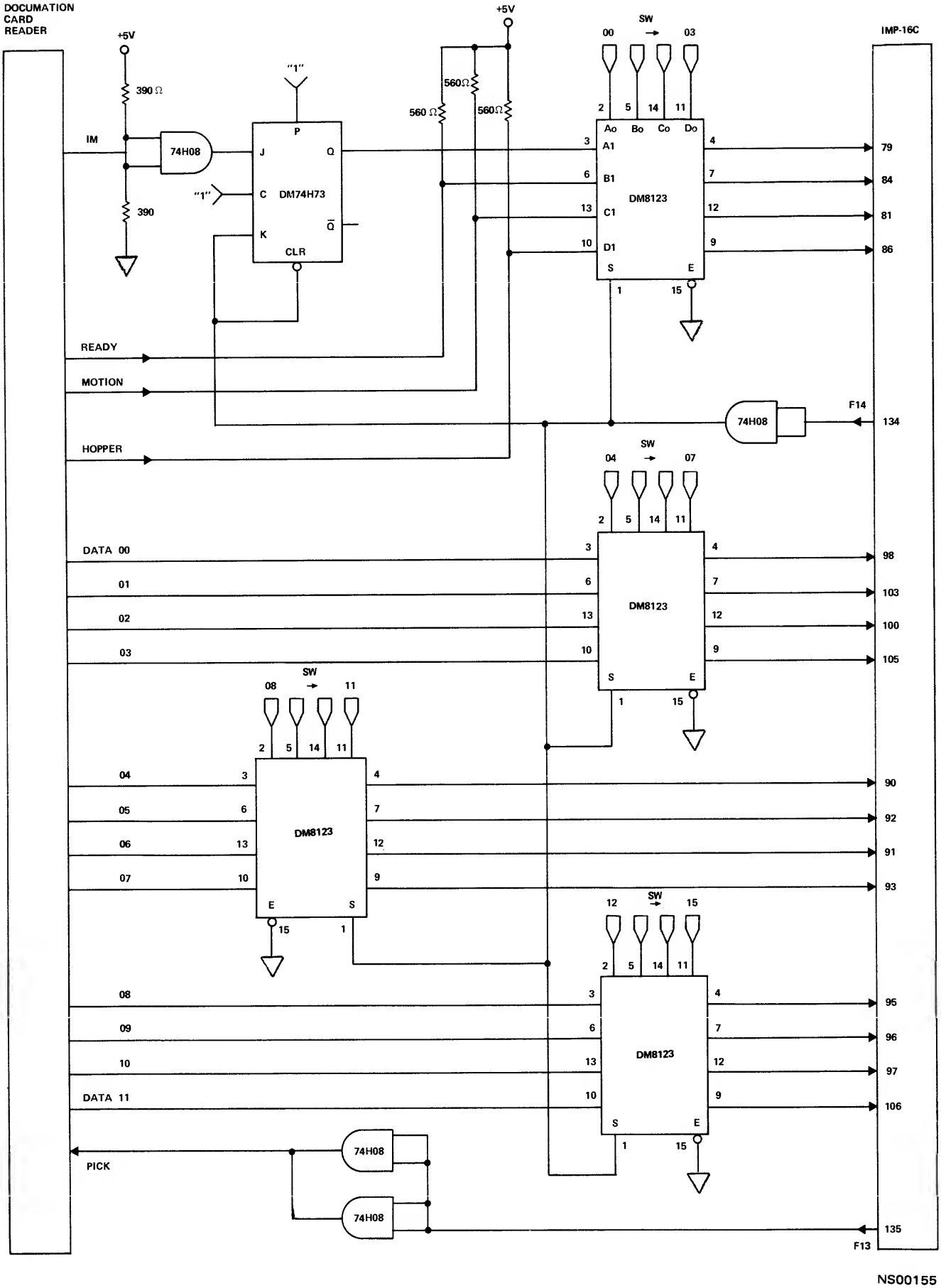


Figure 4-3. Card Reader Interface, Flag-Controlled

```
        BIT0 = 3
        BIT1 = 4
        PICK = 5
        READCARD = 6
        BUFFER = 256 - 80

RDCARD: PUSH    2
$2:     LD      3,ABUF          ; LOAD ADDRESS OF BUFFER.
        LI      2,80            ; SET COLUMN COUNT.
        SFLG    PICK            ; ENABLE PICK FLAG.
$0:     SFLG    READCARD        ; ENABLE DATA INPUT BUFFERS.
        RIN                     ; GET DATA.
        BOC     BIT0,$1         ; TEST INDEX MARK.
        SKAZ    0,$C            ; TEST HOPPER AND
        JMP     INERR           ; MOTION CHECK.
        BOC     BIT1,$0         ; TEST READY.
INERR:  RCPY    R0,R1
        PFLG    PICK
        LI      R0,1
        HALT
        JMP     $2
$C:     .WORD   0C
$1:     PFLG    PICK            ; START COLUMN DATA PROCESSING,
        PFLG    READCARD        ; DISABLE PICK AND INDEX FF.
        SHR     0,4             ; STRIP STATUS BITS.
        ST      0,(3)           ; SAVE CHARACTER IN BUFFER.
        AISZ    3,1             ; INCREMENT BUFFER ADDRESS.
        AISZ    2,-1            ; DECREMENT COLUMN COUNT.
        JMP     $0              ; JUMP TO READ NEXT COLUMN
        PULL    2
ABUF:   .WORD   BUFFER          ; ADDRESS OF BUFFER HERE.
```

4.3 PROGRAM CONTROLLED INTERFACE USING DEVICE ADDRESSES

This approach is a more-conventional approach to I/O interfacing because it makes use of decoded device addresses provided by RIN and ROUT instructions. The number of ICs required increases slightly.

The program listing below describes a read card routine that works with this interface. The hardware is shown in figure 4-4.

Note that this interface does not depend on the use of the simple control panel because a unique device address has been assigned to the card reader. All peripheral devices (including any type of control panel) would communicate with the IMP-16C card via a tri-state input bus connected to the SW lines. In order words, only one device at a time controls the input bus as selected by the appropriate device address. The control panel (switches and lights) would have its own device address.

```
        .PAGE   'READCARD ROUTINE'
        CRADR = 02*8            ; CARD READER ADDRESS
        PICK  = 2               ; PICK COMMAND ORDER CODE.
        RESET = 3               ; RESET PICK FLIP-FLOP.
        READ  = 1               ; READ DATA
RDCARD: PUSH    2
        PUSH    1
        LD      2,ABUF
        LI      3,CRADR         ; LOAD CARD READER ADDRESS.
        LI      1,80
        ROUT    PICK            ; GET CARD
$STRT:  RIN     READ            ; GET DATA
        BOC     BIT1,.+2        ; LOOP UNTIL READY.
        JMP     .-2
        BOC     BIT0,COL        ; TEST FOR INDEX MARK.
        SKAZ    0,$C            ; TEST FOR HOPPER/MOTION CHECK.
        JMP     MOTERR
        JMP     $STRT
COL:    ROUT    RESET
        SHR     0,4             ; STRIP STATUS BITS.
        ST      0,(2)           ; SAVE DATA IN BUFFER
        ADD     2,$ONE          ; INCREMENT BUFFER ADDRESS
        AISZ    1,-1            ; DECREMENT COLUMN COUNT
        JMP     $STRT
        PULL    1
        PULL    2
        RTS
$C:     .WORD   0C
MOTERR: RCPY    0,1
        ROUT    RESET
        LI      0,1
        HALT
ABUF:   .WORD   0               ; ADDRESS OF BUFFER HERE.
$ONE:   .WORD   1
```

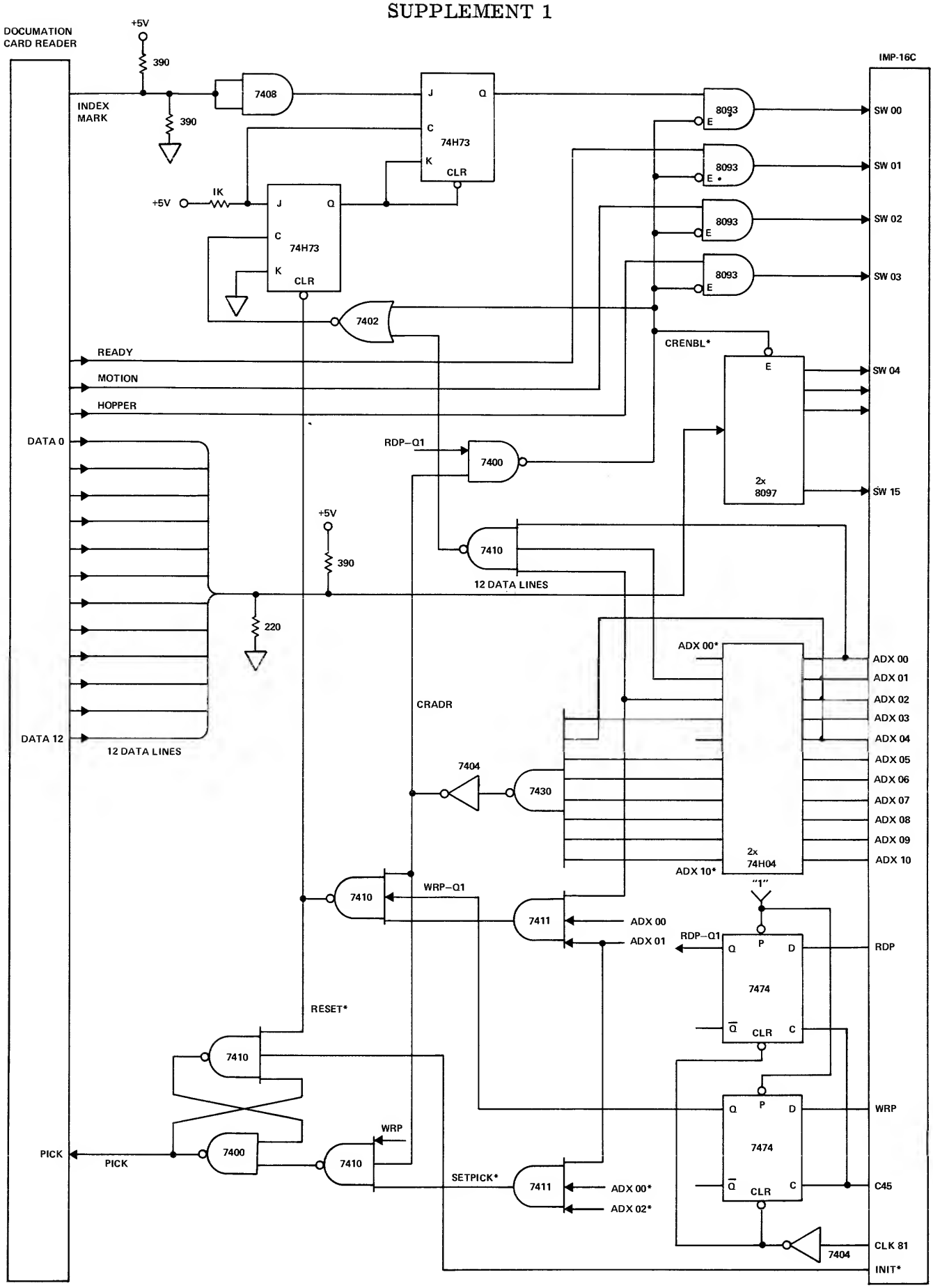


Figure 4-4. Card Reader Interface, Device Address-Controlled

NS00156

CHAPTER 5

LOADER ROUTINES

5.1 GENERAL INFORMATION

The IMP-16 Assembler creates load modules (tapes or cards) from source programs written in assembly language. These load modules (called RLMs) contain the object code for the program to be executed, relocation information, and other loading information, arranged in four kinds of records: a "title" record, one or more "symbol" records, one or more "data" records, and an "end" record. Specific details of the various record formats are given in the IMP-16 Assembler Manual (4200002).

The loader programs described in the following paragraphs may be used with the routines and programs presented in chapters 3 and 4.

5.2 ABSOLUTE LOADER (ABSLDR)

The following loader program takes an absolute paper tape (in RLM format) and loads it into memory from a starting address defined on the tape. After loading the program, the IMP-16C halts and the program entry address is displayed. If EXECUTE is pressed, the processor jumps to the loaded program and begins execution.

This loader program may be used with any of the TTY routines described in chapter 3. That is, either GETC or RECV and PUTC or SEND may be used. The listings shown here call SEND and RECV, but the JSR calls may be changed appropriately to call PUTC and GETC respectively.

```
        .PAGE   'TTYLDR FROM RLM TAPES'
ABSLDR: JSR     RECV
        AISZ    0,-2            ; LOOK FOR STX CHARACTER.
        JMP     ABSLDR
        JSR     RDPACK2         ; START PROCESSING RECORD CONTROL INFO
        BOC     C2,TORS         ; IF BIT15=1 GO TO TITLE, SYMBL PRCSSING
        SHL     0,1
        BOC     C2,.+2          ; IF BIT14=0 GO TO DATA RECORD.
        JMP     ENDREC
DATREC: SHR     0,1
        RCPY    0,3
        JSR     RDPACK2
        RCPY    0,1
        PUSH    0               ; SAVE CHECKSUM WORD.
        CAI     1,1
        JSR     RDCHK
        JSR     RDCHK
        RCPY    0,2
        JSR     RDCHK
        JSR     RDCHK
        RADD    2,3
        CAI     3,1+4
```

(listing continued)

```
LOAD:    JSR     RDCHK
         ST      0,(2)
         ADD     2,$ONE          ; INCREMENT ADDRESS.
         RCPY    2,0
         RADD    3,0
         BOC     C2,CKTEST
         JMP     LOAD
RDCHK:   JSR     RDPACK2
         RADD    0,1
         RTS
CKTEST:  PULL    0               ; CHECKSUM TEST
         BOC     C1,ABSLDR
         RCPY    1,0
         BOC     C1,ABSLDR
         LI      0,X'33
         HALT                    ; CHECKSUM ERROR.
TORS:    AND     0,H3FFF
         RCPY    0,1
         JSR     RDPACK2
         AISZ    1,-1
         JMP     .-2
         JSR     RDPACK2
         JMP     ABSLDR
ENDREC:  JSR     RDPACK2
         JSR     RDPACK2
         JSR     RDPACK2
         HALT
RDPACK2:JSR      RECV            ; READS AND PACKS TWO CHAR IN AC0.
         PUSH    1
         SHL     0,8
         PUSH    0
         JSR     RECV
         PULL    1
         RXOR    1,0
         PULL    1
         RTS
```

5.3 PAPER TAPE BOOTSTRAP LOADER (PTBOOT)

For programs of short length, it is sometimes desirable to load directly into memory without going through an assembly. For such cases, the following program serves the purpose of loading hexadecimal information directly from paper tape. The loader format is laid out such that the first 4 hexadecimal characters (that is, 4 rows of 8-bit ASCII coded characters) from the paper tape are interpreted as the starting address. All subsequent blocks of 4 characters each are interpreted as data and packed into 16-bit words. The last recognizable character on the tape should be "!". All data characters are arranged so that the most significant bits appear first. The PTBOOT program reads the tape and returns control to the panel routine. This assumes that this program is used with the control panel routine described in chapter 2. PTBOOT calls the teletype RECV routine described in 3.2.2.

PTBOOT PROGRAM

```
LOOP1:  LI 1,4
        JSR PTBOOT
        RCPY 2,3;               READ FIRST 4 WORDS; THIS IS THE
;                               STARTING ADDRESS.
LOOP2:  LI 1,4;
        JSR PTBOOT
        ST 2,(3);               READ AND STORE 4 HEX WORDS.
        ADD 3,ONE;              INCREMENT MEMORY ADDRESS.
        JMP LOOP2
        LI 0,X'77;              ERROR CODE X'77
        HALT;                   ATTEMPTED TO LOAD LOC 0.
PTBOOT: JSR RECV;               GET ONE WORD.
PACK:   AND 0,MSKPAR;           MASK OUT PARITY BIT.
        SKNE 0,EXCLAM;
        JMP $OUT
        SKNE 0,CRETRN;          IGNORE  CARRIAGE RETURN.
        JMP PTBOOT
        SKNE 0,LINEFD;          IGNORE LINEFEED.
        JMP PTBOOT
        SKAZ 0,NUMBER;
        JMP NUM
        SKAZ 0,ALPHA;
        JMP ALFA
        LI 0,3;                 ERROR CODE: INVALID CHARACTER.
        JMP 0;
ALFA:   ADD 0,NINE;
NUM:    AND 0,MSK1
        SHL 2,4
        RXOR 0,2
        AISZ 1,-1
        JMP PTBOOT
        RTS
$OUT:   PULL 1
        PULL 1
        HALT
NINE:   .WORD 9
MSKPAR: .WORD 07F
MSK1:   .WORD 0F
NUMBER: .WORD 030
ALPHA:  .WORD 040
CRETRN: .WORD 0D
LINEFD: .WORD 0A
EXCLAM: .WORD 021
RECV1:  .WORD RECV
```

5.4 CARD READER LOADER (CRLM)

This loader program takes an absolute RLM in card format and loads it into memory. It uses 80 words of memory as a temporary buffer. For the example program given here, locations 176 to 255 in base page are used for the buffer. The object deck to be loaded must be followed by a !G card. This loader routine calls a RDCARD subroutine, which may be any one of the two described in chapter 4.

```
;               THIS IS AN ABSOLUTE LOADER FOR THE IMP-16C/P SYSTEM
;
;               ( BASE PAGE VERSION ).
;
;               THIS PROGRAM READS ONE RLM FROM THE CARD READER AND
;               LOADS IT INTO MEMORY.  THE RLM MUST HAVE BEEN PUNCHED
;               INTO CARD COLUMNS 1-72, AND CAN CONTAIN PUNCH CODES
;               ONLY FOR THE CHARACTERS 0,1,...,9,A,B,...,F, OR BLANK.
;               BLANKS WILL BE TREATED AS 0.
;
;               THE RLM MUST BE IN STANDARD RLM FORMAT. THE TITLE CARD
;               AND SYMBOL CARDS ARE IGNORED. DATA FROM DATA CARDS IS
;               MOVED TO THE SPECIFIED LOAD LOCATIONS WITHOUT ANY
;               RELOCATION PERFORMED.  THE END CARD MUST CONTAIN AN
;               ENTRY POINT ADDRESS (SEE ERROR CODE 5, BELOW) AND IS
;               LAST CARD READ BY ABSCR.
;
;               A CHECKSUM TEST IS PERFORMED ON ALL INPUT CARDS.  (SEE
;               ERROR CODE 3, BELOW.)
;
;               BUFFER AREA IN BASE PAGE, LOCS. 256-80.
;
;
;       ERROR   MEANING             ACTION
;       -----   ----------------    ------------------------------------
;         1     I/O ERROR           REPLACE CARD IN READER AND PUSH START
;         2     INV. CHARACTER      CORRECT CARD, REPLACE IN READER, AND
;                                      PUSH START.  (ONLY CODES 0,...,F
;                                      AND BLANK ARE ALLOWED.)
;         3     CHECKSUM ERROR      CORRECT CARD, REPLACE IN READER, AND
;                                      PUSH START.
;         5     INV. ENTRY POINT    SET CORRECT ENTRY POINT INTO REG. 1
;                                      AND PUSH START.
;
;       ALL ERROR CODES ARE LOADED INTO REG. 0 BEFORE HALTING.


        .PAGE   'ABSCR ROUTINE - IMP-16C'
R0 = 0
R1 = 1
R2 = 2
R3 = 3
BIT0 = 3
BIT1 = 4
NZERO= 5
```

```
INLOOP: JSR  RDCARD;          READ ONE CARD.
        JSR  CNVRT;           PACK DATA.
        LD 3,ABUF;
        LD 0,(3)
        AND 0,MASK3
        RCPY 0,2
        LI 1,0;               SUM=0
        SKNE 1,1(3);          IF CHECKSUM = 0, DATA VALID.
        JMP VALID
CKSUML: ADD 1,2(3)
        AISZ 3,1;             INCREMENT ADDRESS
        AISZ 2,-1;
        JMP CKSUML
        LD 3,ABUF
        SKNE 1,1(3);          COMPARE SUM&CKSUM.
        JMP VALID
ERR3:   LI 0,3;               ERROR CODE 3; CHECKSUM ERROR
        HALT
        JMP INLOOP
VALID:  LD 2,(3)
        SHR 2,14;             ISOLATE CODE FOR RECORD TYPE.
        ADD 2,JTBL            AC2 <- CODE + JUMP TABLE ADDRESS.
        JMP @(2);             JUMP TO PROCESS RECORD TYPE.



TITLE:  JMP INLOOP;           IGNORE TITLE CARD.



SYMBOL: JMP INLOOP;           IGNORE SYMBOL CARD.



END:    LD 1,3(3);             SAVE PROGRAM ENTRY POINT
        ST 1,8;               LOCATION 8 HAS ENTRY POINT
        JMP INLOOP
```

(listing continued)

```
EBUF:    .WORD   BUFFER+72;
BUFFER   = 256 - 80
ATBL:    .WORD   TBL;
ETBL:    .WORD   TBL+15;
JTBL:    .WORD   JUMPTBL;
JUMPTBL:.WORD    TITLE,SYMBOL,DATA,END;
DATA:    LD      R2,3(R3);       A <-- INITIAL LOAD ADDRESS
         AISZ    R0,-4;          L <-- L - 4 (SKIP OVER RELOC. INFO)
         AISZ    R3,6;           P <-- P + 6 (SKIP OVER RELOC. INFO)
DLOOP:   LD      R1,0(R3);
         ST      R1,0(R2);       WORD(A) <-- WORD(P)
         AISZ    R3,1;           P <-- P + 1
         AISZ    R2,1;           A <-- A + 1
         AISZ    R0,-1;          L <-- L - 1, SKIP IF DONE
         JMP     DLOOP;          LOOP FOR NEXT WORD
         JMP     INLOOP;         LOOP FOR NEXT CARD

         .PAGE   'HOLLERITH TO BINARY*CONVERSION'
         ENTRY = 8;              ENTRY POINT IN LOCATION 8.
CNVRT: LD 3,ABUF;                FETCH BUFFER ADDRESS.
         RCPY 3,2
         LD 1,(3)
         SKNE 1,EXCLAM;          IF "!G" CARD, GO TO ENTRY POINT
         JMP .+2
         JMP LOOP1
         LD 1,1(3)
         SKNE 1,G
         JMP .+2
         JMP LOOP1
         PULL 0;                 EXIT FROM SUBROUTINE; POP STACK
         LD 2,ENTRY
         SKG 2,ZERO;             IF ENTRY POINT >0, SKIP
         JMP ERROR5;             ELSE ERROR CODE 5.
$EX:      LI 3,1;                IDENTIFY LOAD DEVICE AS CARD READER
         JMP (2);                JUMP TO LOADED PROGRAM
ERROR5: LI 0,5;                  ERROR 5: INVALID ENTRY POINT.
ZERO:    HALT
         RCPY 1,2;               HALT AND THEN
         JMP $EX;                JUMP TO LOADED PROGRAM.


LOOP1:   LI 0,-4
         LD 1,(3)
         ST 3,7;                 SAVE CURRENT BUFFER LOCATION
         LD 3,ATBL
LOOP2:   SKNE 1,(3)
         JMP STEP2
         ADD 3,TBL+9;            INCREMENT ADDRESS
         SKG 3,ETBL;             SKIP IF END OF TABLE
         JMP LOOP2
         SKNE 1,ZERO;            IF NOT BLANK THEN SKIP.
         JMP BLANK
ERR2:    LI 0,2;                 INVALID CHARACTER.
         HALT;                   HALT AND THEN
         PULL 0;                 REREAD CARD.
         JMP INLOOP
```

```
BLANK:  LD 3,ATBL
STEP2:  SUB 3,ATBL
        LD 1,9
        SHL 1,4
        RXOR 3,1;               MERGE 4 BITS AT A TIME.
        ST 1,9;                 SAVE PARTIALLY PACKED WORD.
        LD 3,7;                 GET BUFFER LOCATION
        ADD 3,TBL+9;            INCREMENT BUFFER LOCATION
        AISZ 0,1;               CHECK TO SEE IF 4 WORDS COLLECTED.
        JMP LOOP1+1
        ST 1,(2);               STORE PACKED WORD
        ADD 2,TBL+9
        SKNE 3,EBUF
        RTS
        JMP LOOP1



MASK1:  .WORD   X'C0            ; TRANSMISSION ERROR OR DATA OVERRUN.
MASK3:  .WORD   X'3FFF          ; ALL BITS BUT RECORD TYPE.
THREE:  .WORD   3
EXCLAM: .WORD   X'482
G:      .WORD   X'804
TBL:    .WORD   0200,0100,0080,0040,0020,0010
MASK2:  .WORD   0008            ; HOLLERITH 6 ALSO ERROR CODE FOR CR BUSY.
        .WORD   0004,0002,0001
        .WORD   0900,0880,0840,0820,0810,0808
        .END    INLOOP
```

CHAPTER 6

APPLICATION PROGRAMS

6.1 GENERAL INFORMATION

This chapter presents two application programs that can be used to obtain memory dumps onto paper tapes as described in the following paragraphs.

6.2 PROM TAPE GENERATOR

This program takes the contents of a specified range of memory locations and dumps them on to paper tape in binary 8-channel format suitable for programming ROMs on the DATA I/O PROM programmer. The first tape generated contains the left byte of the 16-bit word and the second tape contains the right byte. This routine calls the SEND routine of the teletype utilities package. Before executing the program, AC2 should contain the starting address of the range and AC3 the final address. The DATA I/O machine has negative logic, so the data bits are complemented before being punched.

```
; PROM TAPE PROGRAM
; -----------------
BLNK:   LI      1,20
        LI      0,-1
        JSR@    SEND1           ; ROUTINE TO PROVIDE
        AISZ    1,-1            ; LEADING BLANKS
        JMP     .-2
        RTS
        JSR     BLNK
        PUSH    2               ; SAVE STARTING ADDRESS
        LD      1,PROM          ; DUMMY WORD FOR COMPARISON
PROM:   LD      0,(2)
        CAI     0,0             ; INVERT BITS FOR DATA I/O
                                ; MACHINE
        SKNE    1,PROM          ; SKIP FOR RIGHT SIDE
        SHR     0,8             ; RIGHT JUSTIFY MSBYTE
        JSR@    SEND1
        RCPY    3,0             ; CHECK TO SEE IF DONE
        RXOR    2,0
        BOC     1,.+3           ; IF DONE GO TO HALT
        AISZ    2,1             ; INCREMENT ADDRESS
        JMP     PROM
        HALT                    ; AWAIT "EXECUTE" FOR
                                ; RIGHT SIDE
        JSR     BLNK
        PULL    2               ; RESTORE ADDRESS
        JMP     PROM            ; GENERATE RIGHT SIDE TAPE
SEND1:  .WORD   0FF53           ; ADDRESS OF SEND ROUTINE IN CUTIL
```

6.3 PAPER TAPE PUNCH PROGRAM

This program generates an absolute paper tape from the contents of a specified range in memory. The first four characters on the tape are the ASCII equivalent of the four hexadecimal numbers that specify the starting address. The last character is an exclamation mark; this serves as a termination indicator. The listing for this program is given in appendix A.

CHAPTER 7

INTERRUPT HANDLING

7.1 GENERAL INFORMATION

In the IMP-16C, there are two processor interrupt request lines; one of these is reserved for stack overflow interrupts. All external peripheral devices are wired to the main interrupt request line (INTRA). If any device generates an interrupt request, the line goes high and interrupts the processor if the master interrupt enable (INTEN) is set for the processor.

7.2 INTERRUPT RESPONSE

Response to a processor interrupt occurs at the end of the instruction executing at the time the interrupt occurs. The interrupt causes the processor to save the current state of the program counter (PC) on the top of the stack and to set the new contents of the PC equal to 1. The interrupt enable flag (INTEN) is then turned off, and the processor executes the instruction in memory location 1, which is the start of the interrupt service routine. The interrupt service routine may determine the presence of a stack overflow interrupt by use of the BOC instruction. The instruction in location 1 may also be a jump to an interrupt routine located elsewhere.

Available at the card-edge connector of the IMP-16C are two status flags (Flag 0 and Flag 12); these status flags may be used in multi-level interrupt schemes. These flags would then serve as interrupt enable signals for each level of a two-level system. Figure 7-1 shows the external circuits required for this function.

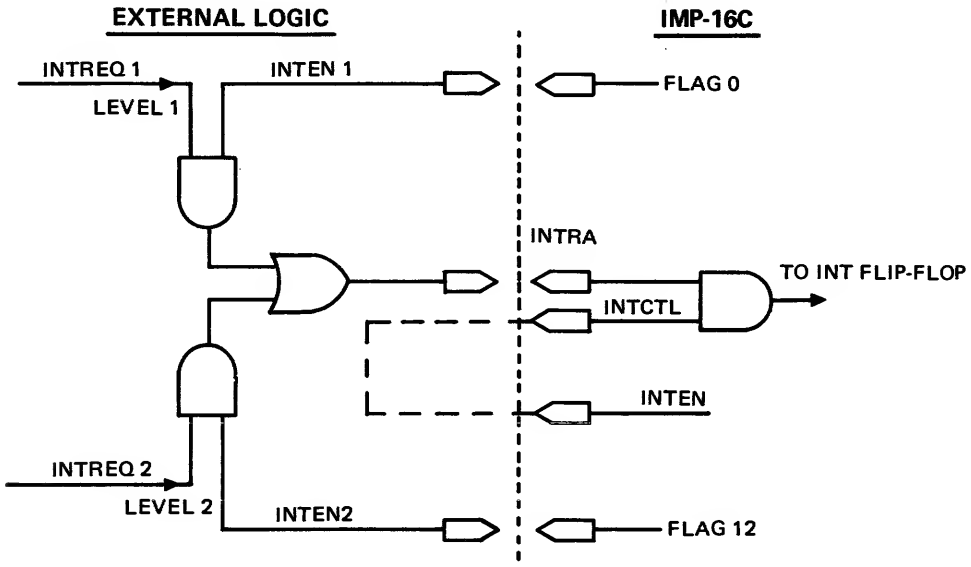


Figure 7-1. Interrupt Response External Circuits

7.3 INTERRUPT GENERATION AND PROCESSING

The following discussion pertains to a two-level interrupt system, but all the operations apply to single-level interrupts as well.

7.3.1 General Interrupt

Interrupt requests on levels 1 and 2 require that the interrupt service routine determine the address of the interrupting device. This can be done conveniently using the "Interrupt Select Status" order as described below.

A peripheral controller requiring interrupt servicing by the processor sets its interrupt request flag, thus causing a true signal on the interrupt request line if the peripheral's interrupt enable flag is set true. The interrupt request line is common to all peripheral controllers on a given level and only requires that one peripheral device be requesting service in order to set the line to the true state. The main program, when ready, sets the processor INTEN flip-flop, thus indicating that the processor may perform an interrupt service. If both the processor INTEN and the interrupt enable for the given level are enabled at the same time, the processor transfers program control to the interrupt service routine.

When the processor responds to an interrupt request, it goes through the following procedure in order to transfer program control to the interrupt service routine:

1. Transfers the contents of the Program Counter (PC) to the top of the stack.

2. Places the address of memory location 1 into the PC.

3. Disables (clears) the processor INTEN flag to prevent further interrupt.

4. Fetches the next instruction from memory location 1, thus initiating the interrupt service routine.

When an interrupt occurs, the interrupt service routine directs the processor to determine the devices requiring service and to select one peripheral device. This may be accomplished as follows:

1. An "interrupt select status" order is sent out to all peripheral controllers. The order field of the command word is the only field recognized by the peripheral controllers, and the address field is ignored. Upon receipt of the "interrupt select status" order, the INT REQ flags at the peripheral devices are cleared in most cases.

2. Each peripheral device is assigned one of the 16 system bus lines to report its interrupt status. Each peripheral device responds simultaneously with other peripheral devices, indicating whether or not it requires interrupt servicing. A binary 1 indicates a service request. Typical interrupt assignments are shown in table 7-1.

3. The interrupt service routine resolves interrupt priority and selects the peripheral device for interrupt servicing.

NOTE

Although there are only 16 system bus lines that are used for reporting interrupt status, by use of two "interrupt select status" orders, 1 and 2, the status of 32 peripheral devices may be determined, one group of 16 peripheral devices responding to "interrupt select status 1" and another group to "interrupt select status 2." This concept can be extended to more than two levels of interrupt select status orders if necessary.

After a peripheral device obtains interrupt access, the applicable RIN or ROUT command then effects the transfer of data between the processor and the peripheral controller. Upon completion of the interrupt operations, program control is transferred back to the main program by use of the RTI instruction. The RTI instruction causes the program counter to be loaded by adding the top word of the stack to the CTL field of the instruction. The INTEN flip-flop is then enabled.

Table 7-1. Typical Interrupt Select Status 1 Bit Assignments

Bit	Assigned Peripheral
0	Unassigned
1	Parallel Teletype
2	Card Reader
3	Disc
4	Communications
5	Interval Timer
6	Unassigned
7	Serial Teletype
8	Modem
9	Unassigned
10	Unassigned

(table continued)

Table 7-1. Typical Interrupt Select Status 1 Bit Assignment (Cont.)

Bit	Assigned Peripheral
11	Unassigned
12	Unassigned
13	Unassigned
14	Unassigned
15	Unassigned

7.3.2 Stack Overflow Interrupt

If the processor stack becomes full, a stack overflow interrupt is set. If the INTEN flag is set, then the processor is interrupted. The interrupt service routine may then test jump condition 8 to determine that the interrupt was caused by a stack overflow.

In most applications, the software may be written in a manner that will guarantee that a stack overflow will not occur. However, even in such cases, it is possible in the system programming development that a stack overflow may accidentally occur. Because the interrupt process itself utilizes the stack, some words of the stack may be lost in the overflow interrupting sequence.

The systems programmer must take some precautions to guarantee that a stack overflow error does not go undetected and that data on the stack are not lost. The programmer must not push words of all zeros onto the stack as data; two protection words of all ones should be kept at the bottom of the stack. These are relatively minor restrictions since, in most applications, use of the stack is reserved for subroutine return addresses, interrupt return addresses, and for saving the RALU flags while servicing interrupts.

7.4 SAMPLE PROGRAM FOR INTERRUPT PROCESSING

Consider the case of four peripheral devices connected to the IMP-16C in a priority scheme such that when an interrupt occurs the interrupting device sends over the bus a data word whose value identifies the device. The following program segment is a service routine that determines the identity of the device and sends control to a device routine. After the device has been serviced, the interrupt routine returns control to the main program.

The four devices for this example are a keyboard, an assemblage of digital integrators, a display unit, and a printer unit — arranged such that the keyboard assumes the highest priority. The keyboard indicates its status by sending a high level on data bit 0 of the bus; the digital integrators use bit 1, the display uses bit 2, and the printer uses bit 3. The values returned are, therefore, 1, 2, 4, and 8. The order code for reading status has been chosen arbitrarily as 2.

In this example, the IMP-16C is the host processor that monitors the operation of the peripheral system components. The digital integrators operate independently, requiring processor intervention only to exchange data, initializing information, and setting up other parameters. The display unit provides a continuous visual display of the operation of the integrators, and the printer provides hard copy when necessary.

```
          INTRAD = X'40
          STATUS = 2
          IX2    = 2
          AC0    = 0


IDTABL:  .WORD.                    ; ADDRESS OF SELF.
         .WORD   KEYBD             ; ADDRESS OF KEYBOARD ROUTINE.
         .WORD   DIGINT            ; ADDRESS OF INTEGRATORS ROUTINE.
         .WORD   BADINT            ; ILLEGAL INTERRUPT ADDRESS.
         .WORD   DISPLA            ; ADDRESS OF DISPLAY ROUTINE.
         .WORD   BADINT            ; ILLEGAL INTERRUPT ADDRESS.
         .WORD   BADINT            ; ILLEGAL INTERRUPT ADDRESS.
         .WORD   BADINT            ; ILLEGAL INTERRUPT ADDRESS.
         .WORD   PRINTR            ; ADDRESS OF PRINTER ROUTINE.
;
;        INTERRUPT SERVICE ROUTINE BEGINS HERE
;
INTSERV: PUSHF                     ; SAVE STATUS FLAGS.
         JSR     SAVE1             ; SAVE ACCUMULATORS IN RAM.
         LI      3,INTRAD
         RIN     STATUS            ; DETERMINE INTERRUPTING DEVICE
;                                    BY READING THE INPUT BUS
;                                    FOR DEVICE STATUS BITS.
         RCPY    AC0,IX2           ; ESTABLISH INDEX VALUE.
         ADD     IX2,IDTABL        ; MAKE A LOCAL ADDRESS.
         JSR@    (IX2)             ; JUMP TO DEVICE SERVICE ROUTINE.
         JSR     RSTOR1            ; RETURN HERE AFTER SERVICING
;                                    DEVICE AND RESTORE ACCUMULATORS.
         PULLF                     ; RESTORE STATUS FLAGS.
         RTI     0                 ; RETURN TO INTERRUPTED PROGRAM.
RSTOR1:  LD      0,SVR0
         LD      1,SVR1
         LD      2,SVR2
         LD      3,SVR3
         RTS     0
SAVE1:   ST      0,SVR0
         ST      1,SVR1
         ST      2,SVR2
         ST      3,SVR3
         RTS     0
SVR0:    .WORD   0                 ; STORAGE LOCATIONS FOR
SVR1:    .WORD   0                 ; ACCUMULATORS.
SVR2:    .WORD   0                 ;
SVR3:    .WORD   0
```

(listing continued)

```
KEYBD:  .                       ; SERVICE ROUTINES FOR
        .                       ; THE VARIOUS PERIPHERALS.
        .
DIGINT: .

DISPLA: .

PRINTR: .

BADINT: .
```

APPENDIX A

ASSEMBLY LISTING

An actual assembly listing is presented on the following pages to illustrate some of the topics described in this manual. This program listing consolidates some of the routines described earlier and is a complete firmware package (available in PROM) that can be installed and run on the IMP-16C with the control panel described in chapter 2. The program is assembled at location $FF00_{16}$ and occupies the top page of memory. A few locations in base page read/write memory are used for temporary storage. The following subroutines and procedures are included in the listing.

1. Control Panel Service Routine
2. Panel HALT Interrupt Routine
3. Teletype Receive Character Subroutine
4. Teletype Transmit Character Subroutine
5. Paper Tape Absolute RLM Loader
6. Paper Tape Loader (BOOTSTRAP)
7. Tape Punch Program

The control panel described in chapter 2 is required for this program package. Additional interface hardware required are the circuits depicted in figure 3-1.

This package is available in ROM under the part name CUTIL. The two teletype routines in this package (SEND and RECV) are written as subroutines so they may be called from user programs with the JSR@ instruction. Entry points and operating instructions are given in the program listings.

```
REVISION-C 11/20/72                                        73227   09143181
CUTIL    CONTROL PANEL AND TTY UTILITIES                   PAGE NUMBER    1


    1 0000                  .TITLE CUTIL,  'CONTROL PANEL AND TTY UTILITIES'
    2 0000                  .ASECT
    3 0000 0001 A           C1 = 1          ;  AC0 = 0
    4 0000 0002 A           C2 = 2          ;  BIT 15 OF AC0 IS 0
    5 0000 0003 A           C3 = 3          ;  BIT 0 OF AC0 IS 1
    6 0000 0004 A           C4 = 4          ;  BIT 1 OF AC0 = 1
    7 0000 0005 A           C5 = 5          ;  AC0 -= 0
    8 0000 0007 A           C7 = 7          ;  START CONDITION
    9 0000 0008 A           C8 = 8          ;  STACK FULL CONDITION
   10 0000 0008 A           C11 = 11        ;  AC0 LTEQ 0
   11 0000 000C A           C12 = 12        ;  JUMP COND.12 (WIRED TO "LOAD ADDRESS")
   12 0000 000D A           C13 = 13        ;  JUMP COND. 13 (WIRED TO " LOAD DATA")
   13 0000 000E A           C14 = 14        ;  JUMP COND. 14
   14 0000 000F A           C15 = 15        ;  JUMP COND. 15 (WIRED TO "DISPLAY")
   15 0000 FF00 A           .=X'FF00
   16 FF00          ;
   17 FF00          ;           THIS UTILITIES PACKAGE CONTAINS A SIMPLE
   18 FF00          ;           CONTROL PANEL ROUTINE THAT OPERATES IN
   19 FF00          ;           CONJUNCTION WITH THE CONTROL PANEL KIT
   20 FF00          ;           (CTLPLKIT),AND SOME TELETYPE ROUTINES
   21 FF00          ;           TO LOAD PROGRAMS AND PUNCH TAPES.
   22 FF00          ;


ABSTTY FOR THE IMP-16C

   23 FF00                  .PAGE   'ABSTTY FOR THE IMP-16C'
   24 FF00          ;
   25 FF00          ;               LOADS ABSOLUTE PAPER TAPE IN RLM FORMAT.
   26 FF00          ;               EACH RECORD MUST BE PRECEDED BY STX.
   27 FF00          ;
   28 FF00          ;               ENTRY POINT FOR THIS PROGRAM IS:  FF00
   29 FF00          ;
   30 FF00 293C A ABSTTY:  JSR RECV
   31 FF01 48FE A          AISZ 0,-2;              LOOK FOR START OF TEXT.
   32 FF02 21FD A          JMP ABSTTY
   33 FF03 292B A TTY1:    JSR RDPCK;              PROCESS RECORD CONTROL INFORMATION.
   34 FF04 121F A          BOC 2,TORS;             BRANCH IF TITLE OR SYMBOL RECORD.
   35 FF05 5C01 A          SHL 0,1
   36 FF06 1201 A          BOC 2,.+2;              BRANCH TO DATA RECORD,
   37 FF07 2123 A          JMP ENDREC;                 ELSE GO TO END RECORD.
   38 FF08 5CFF A          SHR 0,1
   39 FF09 3381 A          RCPY 0,3;               RECORD BODY LENGTH IN AC3.
   40 FF0A 2924 A          JSR RDPCK
   41 FF0B 3181 A          RCPY 0,1;               SAVE CHECKSUM.
   42 FF0C 4000 A          PUSH 0;
   43 FF0D 5101 A          CAI 1,1;                AC1 HAS -(CKSUM MODE WORD).
   44 FF0E 290D A          JSR RDWDCK;             SLOUGH ADDRESS MODE.
   45 FF0F 290C A          JSR RDWDCK;             GET LOAD ADDRESS.
   46 FF10 3281 A          RCPY 0,2;               LOAD ADDRESS IN AC2.
   47 FF11 290A A          JSR RDWDCK;             SLOUGH RELOCATION MODE WORDS.
   48 FF12 2909 A          JSR RDWDCK;
   49 FF13 3800 A          RADD 2,3
   50 FF14 5305 A          CAI 3,1+4;              AC3 HAS -(LAST ADDRESS - 1).
   51 FF15 2906 A TTY2:    JSR RDWDCK;             GET DATA WORD.
   52 FF16 A200 A          ST 0,(2)
   53 FF17 4A01 A          AISZ 2,1;               INCREMENT DESTINATION ADDRESS.
   54 FF18 3881 A          RCPY 2,0
   55 FF19 3C00 A          RADD 3,0
   56 FF1A 1204 A          BOC 2,TSTCKSUM;         IF DONE TEST CHECKSUM.
   57 FF1B 21F9 A          JMP TTY2
   58 FF1C 2912 A RDWDCK:  JSR RDPCK
   59 FF1D 3100 A          RADD 0,1
   60 FF1E 0200 A          RTS
   61 FF1F 4400 A TSTCKSUM: PULL 0;                GET CHECKSUM WORD.
```

```
REVISION-C 11/20/72                                          73227   09143181
CUTIL    CONTROL PANEL AND TTY UTILITIES                     PAGE NUMBER    2
   62 FF20 11DF A          BOC C1,ABSTTY
   63 FF21 3481 A          RCPY 1,0
   64 FF22 11DD A          BOC C1,ABSTTY
   65 FF23 0000 A          HALT;                    CHECKSUM ERROR.
   66 FF24 6117 A TORS:    AND 0,H3FFF;             IGNORE TITLE AND SYMBOL RECORDS.
   67 FF25 3181 A          RCPY 0,1
   68 FF26 2908 A          JSR RDPCK
   69 FF27 49FF A          AISZ 1,-1
   70 FF28 21FD A          JMP .-2
   71 FF29 2905 A          JSR RDPCK
   72 FF2A 21D5 A          JMP ABSTTY
   73 FF2B 2903 A ENDREC:  JSR RDPCK;               SLOUGH CHECKSUM.
   74 FF2C 2902 A          JSR RDPCK;               SLOUGH ENTRY ADDRESS MODE.

   75 FF2D 2901 A          JSR RDPCK;               GET ENTRY ADDRESS
   76 FF2E 2000 A          JMP 0;                   JUMP TO PANEL ROUTINE.

   77 FF2F                 .SPACE 3
   78 FF2F 4100 A RDPCK:   PUSH 1
   79 FF30 290C A          JSR RECV
   80 FF31 5C08 A          SHL 0,8
   81 FF32 4000 A          PUSH 0
   82 FF33 2909 A          JSR RECV
   83 FF34 4500 A          PULL 1
   84 FF35 3482 A          RXOR 1,0
   85 FF36 4500 A EXIT11:  PULL 1
   86 FF37 0200 A          RTS
   87 FF38 852E A DELAY:   LD 1,V2;                 DELAY SUBROUTINE
   88 FF39 49FF A          AISZ 1,-1
   89 FF3A 21FE A          JMP .-1
   90 FF3B 0200 A          RTS
   91 FF3C 3FFF A H3FFF:   .WORD X'3FFF


TELETYPE GET CHARACTER ROUTINE

   92 FF3D                 .PAGE   'TELETYPE GET CHARACTER ROUTINE'
   93 FF3D 000E A          JC14 = 14;               INPUT JUMP CONDITION.
   94 FF3D 0003 A          READR = 3;               READER ENABLE FLAG.
   95 FF3D 0004 A          XMIT = 4;                TTY TRANSMIT FLAG.
   96 FF3D 0001 A          C1 = 1
   97 FF3D 0002 A          C2 = 2
   98 FF3D 4100 A RECV:    PUSH 1;                  SAVE ACCUMULATORS.
   99 FF3E 4200 A          PUSH 2
  100 FF3F 0A80 A          PFLG 2;                  DISABLE LINK.
  101 FF40 4E08 A          LI 2,8;                  SET COUNT FOR 8 BITS.
  102 FF41 0B00 A          SFLG READR
  103 FF42 1E01 A          BOC JC14,.+2;            TEST FOR START BIT.
  104 FF43 21FE A          JMP .-1;                 LOOP UNTIL FOUND.
  105 FF44 8521 A          LD 1,V1;                 LOAD TIMING PARAMETER.
  106 FF45 29F3 A          JSR DELAY+1;             DELAY HALF BIT TIME.
  107 FF46 0B80 A          PFLG READR;
  108 FF47 1E01 A          BOC JC14,.+2;            TEST FOR DATA BIT.
  109 FF48 21F6 A          JMP RECV+2
  110 FF49 29EE A REP:     JSR DELAY;
  111 FF4A 5CFF A          SHR 0,1;
  112 FF4B 1E01 A          BOC JC14,.+2;
  113 FF4C 6918 A          OR 0,H8000;
  114 FF4D 4AFF A          AISZ 2,-1;               DECREMENT COUNT.
  115 FF4E 21FA A          JMP REP;
  116 FF4F 29E8 A          JSR DELAY;
  117 FF50 5CF8 A          SHR 0,8;
  118 FF51 4600 A          PULL 2;
  119 FF52 21E3 A          JMP EXIT11;              THIS STEP DONE TO SAVE
  120 FF53        ;                                  PROGRAM STORAGE SPACE.
```

(listing continued)

```
REVISION-C 11/20/72                                          73227   09143181
CUTIL    CONTROL PANEL AND TTY UTILITIES                     PAGE NUMBER    3
TRANSMIT CHARACTER ROUTINE

  121 FF53                    .PAGE     'TRANSMIT CHARACTER ROUTINE'
  122 FF53 4200 A SEND:       PUSH 2;                  SAVE ACCUMULATORS.
  123 FF54 4100 A             PUSH 1;
  124 FF55 0C00 A             SFLG XMIT;               SEND START BIT.
  125 FF56 29E1 A             JSR DELAY;
  126 FF57 4E08 A             LI 2,8;                  SET BIT COUNT.
  127 FF58 0C80 A PUT:        PFLG XMIT;
  128 FF59 1301 A             BDC 3,$XX
  129 FF5A 0C00 A             SFLG XMIT;
  130 FF5B 29DC A $XX:        JSR DELAY;               SEND DATA BIT.
  131 FF5C 5CFF A             SHR 0,1;
  132 FF5D 4AFF A             AISZ 2,-1;               TEST TO SEE IF DONE.
  133 FF5E 21F9 A             JMP PUT
  134 FF5F 0C80 A             PFLG XMIT;               SEND TWO STOP BITS.
  135 FF60 29D7 A             JSR DELAY;
  136 FF61 29D6 A             JSR DELAY;
  137 FF62 4500 A             PULL 1
  138 FF63 4600 A             PULL 2
  139 FF64 0200 A             RTS
  140 FF65 8000 A H8000:      .WORD X'8000
  141 FF66 01B1 A V1:         .WORD X'01B1
  142 FF67 035E A V2:         .WORD X'035E

ABSPT

  143 FF68                    .PAGE    'ABSPT'
  144 FF68          ;
  145 FF68          ;                    THIS IS A PAPER TAPE BOOTSTRAP ROUTINE THAT
  146 FF68          ;                    READS 8 CHANNEL TAPE. THE FIRST 4 WORDS
  147 FF68          ;                    DENOTE THE STARTING ADDRESS FOR THE OBJECT
  148 FF68          ;                    PROGRAM BEING LOADED. THE LAST CHARACTER
  149 FF68          ;                    ON THE TAPE MUST BE AN EXCLAMATION MARK.
  150 FF68          ;
  151 FF68          ;                    ENTRY POINT FOR THIS PROGRAM:  FF68
  152 FF68          ;
  153 FF68 4D04 A LOOP1:      LI 1,4
  154 FF69 2906 A             JSR PTBOOT
  155 FF6A 3881 A             RCPY 2,3;                READ FIRST 4 WORDS; THIS IS THE
  156 FF6B          ;                                  STARTING ADDRESS.
  157 FF6B 4D04 A LOOP2:      LI 1,4;
  158 FF6C 2903 A             JSR PTBOOT
  159 FF6D AB00 A             ST 2,(3);                READ AND STORE 4 HEX WORDS.
  160 FF6E CD45 A             ADD 3,ONE;               INCREMENT MEMORY ADDRESS.
  161 FF6F 21FB A             JMP LOOP2
  162 FF70 29CC A PTBOOT:     JSR RECV;                GET ONE WORD.
  163 FF71 6117 A PACK:       AND 0,MSKPAR;            MASK OUT PARITY BIT.
  164 FF72 F11C A             SKNE 0,EXCLAM;
  165 FF73 2111 A             JMP $OUT
  166 FF74 F118 A             SKNE 0,CRETRN;           IGNORE  CARRIAGE RETURN.
  167 FF75 21FA A             JMP PTBOOT
  168 FF76 F117 A             SKNE 0,LINEFD;           IGNORE LINEFEED.
  169 FF77 21F8 A             JMP PTBOOT
  170 FF78 7112 A             SKAZ 0,NUMBER;
  171 FF79 2105 A             JMP NUM
  172 FF7A 7111 A             SKAZ 0,ALPHA;
  173 FF7B 2102 A             JMP ALFA
  174 FF7C 4C03 A             LI 0,3;                  ERROR CODE: INVALID CHARACTER.
  175 FF7D 2000 A             JMP 0;
  176 FF7E C109 A ALFA:       ADD 0,NINE;
  177 FF7F 610A A NUM:        AND 0,MSK1
  178 FF80 5E04 A             SHL 2,4
  179 FF81 3282 A             RXOR 0,2
  180 FF82 49FF A             AISZ 1,-1
  181 FF83 21EC A             JMP PTBOOT
```

```
  182 FF84 0200 A          RTS
  183 FF85 4500 A $OUT:    PULL 1
  184 FF86 4500 A          PULL 1
  185 FF87 2131 A          JMP START;
  186 FF88 0009 A NINE:    .WORD 9
  187 FF89 007F A MSKPAR:  .WORD 07F
  188 FF8A 000F A MSK1:    .WORD 0F
  189 FF8B 0030 A NUMBER:  .WORD 030
  190 FF8C 0040 A ALPHA:   .WORD 040
  191 FF8D 000D A CRETRN:  .WORD 0D
  192 FF8E 000A A LINEFD:  .WORD 0A
  193 FF8F 0021 A EXCLAM:  .WORD 021
```

ASCII TAPE PUNCH ROUTINE

```
  194 FF90                 .PAGE     'ASCII TAPE PUNCH ROUTINE'
  195 FF90         ;
  196 FF90         ;                  THIS PROGRAM PUNCHES OUT ON PAPER TAPE
  197 FF90         ;                  THE CONTENTS OF A SPECIFIED RANGE OF
  198 FF90         ;                  MEMORY LOCATIONS. THE FIRST 4 WORDS ON
  199 FF90         ;                  THE TAPE  ARE THE ADDRESS OF THE STARTING
  200 FF90         ;                  LOCATION. THE LAST WORD ON THE TAPE IS
  201 FF90         ;                  AN EXCLAMATION MARK. PAPER TAPE GENERATED
  202 FF90         ;                  BY THIS ROUTINE MAY BE LOADED USING THE
  203 FF90         ;                  PTBOOT ROUTINE. BEFORE EXECUTING THIS
  204 FF90         ;                  PROGRAM, AC2 MUST BE LOADED WITH THE
  205 FF90         ;                  STARTING ADDRESS OF THE RANGE TO BE
  206 FF90         ;                  DUMPED AND AC3 MUST HAVE THE FINAL ADDRESS
  207 FF90         ;                  OF THE RANGE. EACH SET OF 4 WORDS ON THE
  208 FF90         ;                  TAPE ARE SEPARATED BY CARRIAGE RETURN
  209 FF90         ;                  AND LINEFEED CHARACTERS.
  210 FF90         ;
  211 FF90 3881 A PTPNCH: RCPY 2,0;          COPY STARTING ADDRESS INTO AC0.
  212 FF91 C022 A         ADD 3,ONE;
  213 FF92 4D04 A ASC:    LI 1,4;            COUNT FOR 4 HEX CHARACTERS.
  214 FF93 5804 A         ROL 0,4;
  215 FF94 4000 A         PUSH 0;
  216 FF95 61F4 A         AND 0,MSK1;
  217 FF96 E1F1 A         SKG 0,NINE;
  218 FF97 2112 A         JMP $NU
  219 FF98 D1EF A         SUB 0,NINE;
  220 FF99 48C0 A         AISZ 0,-64
  221 FF9A 2988 A OUTT1:  JSR SEND;          TRANSMIT CHARACTER.
  222 FF9B 49FF A         AISZ 1,-1;         LOOP UNTIL 4 CHARACTERS SENT.
  223 FF9C 210B A         JMP RETN
  224 FF9D 3C81 A         RCPY 3,0;          CHECK TO SEE IF END OF
  225 FF9E 3882 A         RXOR 2,0;              MEMORY RANGE.
  226 FF9F 110C A         BOC 1,$EXCL;       END OF DATA.
  227 FFA0 81EC A         LD 0,CRETRN;
  228 FFA1 2981 A         JSR SEND;          TRANSMIT CARRIAGE RETURN.
  229 FFA2 81EB A         LD 0,LINEFD;
  230 FFA3 29AF A         JSR SEND;          TRANSMIT LINEFEED.
  231 FFA4 4400 A         PULL 0;            POP TOP WORD TO CLEAR.
  232 FFA5 8200 A         LD 0,(2);          BEGIN DATA TRANSMISSION.
  233 FFA6 C90D A         ADD 2,ONE;         POINT TO NEXT LOCATION.
  234 FFA7 21EA A         JMP ASC;
  235 FFA8 4400 A RETN:   PULL 0
  236 FFA9 21E9 A         JMP ASC+1
  237 FFAA 4880 A $NU:    AISZ 0,-80;
  238 FFAB 21EE A         JMP OUTT1
  239 FFAC 81E2 A $EXCL:  LD 0,EXCLAM;       TRANSMIT EXCLAMATION MARK.
  240 FFAD 29A5 A         JSR SEND;
  241 FFAE 2000 A         JMP 0;
```

(listing continued)

```
REVISION-C 11/20/72                                          73227   09143181
CUTIL    CONTROL PANEL AND TTY UTILITIES                     PAGE NUMBER    5
CONTROL PANEL ROUTINE

  242 FFAF                    .PAGE     'CONTROL PANEL ROUTINE'
  243 FFAF          ;         CUTIL CONTROL PANEL:   JULY 31, 1973.
  244 FFAF          ;
  245 FFAF          ;                              THIS CONTROL PANEL USES ALL DEVICE
  246 FFAF          ;                              ADDRESSES; THAT IS, NO ADDRESS
  247 FFAF          ;                              DECODING IS ASSUMED, AND THE PROGRAM
  248 FFAF          ;                              WILL RESPOND TO ALL 'RIN' AND 'ROUT'
  249 FFAF          ;                              INSTRUCTIONS.
  250 FFAF FFB1 A             .=X'FFB1
  251 FFB1 21B8 A JSTRT:      .WORD X'21B8
  252 FFB2 21EF A JINTR:      .WORD X'21EF
  253 FFB3 0005 A FIVE:       .WORD   5
  254 FFB4 0001 A ONE:        .WORD   1
  255 FFB5 8DFC A BEGIN:      LD 3,JINTR
  256 FFB6 AC01 A             ST 3,X'01;        LOAD LOCATION 1 WITH JUMP TO INTERRUPT
  257 FFB7 8DF9 A             LD 3,JSTRT;
  258 FFB8 AC00 A             ST 3,X'00;        LOAD LOCATION 0 WITH JUMP TO CONTROL PANEL
  259 FFB9 292F A START:      JSR SAVE        ;  SAVE ACCUMULATORS.
  260 FFBA 0900 A SET:        SFLG 1          ;  ENABLE INTERRUPT SYSTEM.
  261 FFBB 0600 A ROUT:       ROUT 0
  262 FFBC 1C04 A WAIT:       BOC C12,LA      ;  'LOAD ADDRESS' SWITCH.
  263 FFBD 1D0A A             BOC C13,LD      ;  'LOAD DATA' SWITCH.
  264 FFBE 1710 A             BOC C7,EX;        'EXECUTE' SWITCH.
  265 FFBF 1F25 A             BOC C15,DISP    ;  'DISPLAY' SWITCH.
  266 FFC0 21FB A             JMP .-4         ;  RETURN TO WAIT LOOP.
  267 FFC1 1CFF A LA:         BOC C12,LA      ;  CHECK RELEASE.
  268 FFC2 0400 A             RIN 0           ;  READ SWR.
  269 FFC3 3281 A             RCPY 0,2        ;  SAVE ADDRESS IN AC2.
  270 FFC4 3381 A             RCPY 0,3
  271 FFC5 E1ED A             SKG 0,FIVE;       PREVENTS LOADING OF RESERVED LOCATIONS
  272 FFC6 1213 A             BOC C2,RSRVE
  273 FFC7 21F3 A             JMP ROUT
  274 FFC8 1DFF A LD:         BOC C13,LD;       CHECK RELEASE FOR LOAD DATA SWITCH
  275 FFC9 0400 A             RIN 0;            READ SWITCHES
  276 FFCA F934 A             SKNE 2,LAST6;     PREVENTS LOADING LOCATION 6.
  277 FFCB 21F0 A             JMP WAIT
  278 FFCC A200 A             ST 0,(2)        ;  LOAD MEMORY
  279 FFCD C9E6 A             ADD 2,ONE;        INCREMENT ADDRESS
  280 FFCE 21EC A             JMP ROUT
  281 FFCF 17FF A EX:         BOC C7,EX       ;  CHECK RELEASE
  282 FFD0 0400 A             RIN 0
  283 FFD1 4000 A             PUSH 0          ;  SAVE JUMP ADDRESS IN STACK.
  284 FFD2 8406 A RSTOR:      LD 1,X'06
  285 FFD3 4100 A             PUSH 1
  286 FFD4 0280 A             PULLF
  287 FFD5 8002 A             LD 0,X'02       ;   RESTORE ACCUMULATORS.
  288 FFD6 8403 A             LD 1,X'03
  289 FFD7 8804 A             LD 2,X'04
  290 FFD8 8C05 A             LD 3,X'05
  291 FFD9 0200 A             RTS 0           ; FAKING AN INDIRECT JUMP.
  292 FFDA 0600 A RSRVE:      ROUT 0
  293 FFDB 1D05 A             BOC C13,LDAC
  294 FFDC 1F01 A             BOC C15,DISPAC
  295 FFDD 21FD A             JMP .-2
  296 FFDE 1FFF A DISPAC:     BOC C15,DISPAC;   DISPLAY ACCUMULATOR ROUTINE
  297 FFDF 8302 A             LD 0,2(3)
  298 FFE0 21DA A             JMP ROUT
  299 FFE1 1DFF A LDAC:       BOC C13,LDAC
  300 FFE2 0400 A             RIN 0
  301 FFE3 A202 A             ST 0,2(2)
  302 FFE4 21D6 A             JMP ROUT
  303 FFE5 1FFF A DISP:       BOC C15,DISP
  304 FFE6 8300 A             LD 0,(3)
  305 FFE7 CDCC A             ADD 3,ONE;        INCREMENT ADDRESS
```

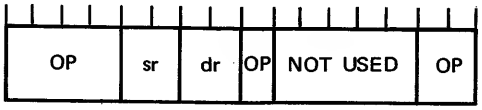
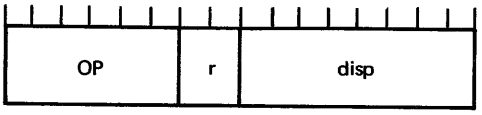
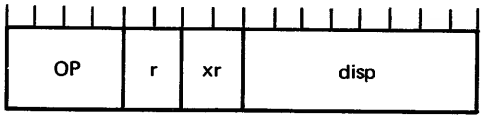
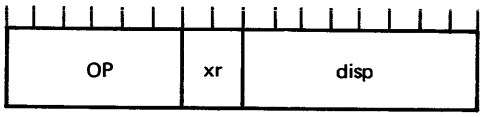
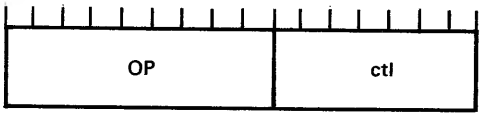
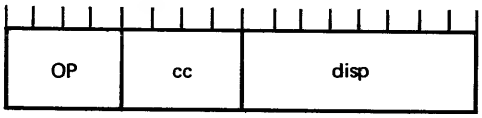
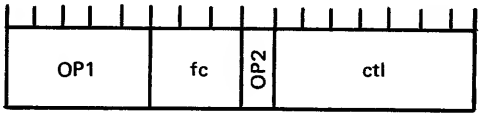
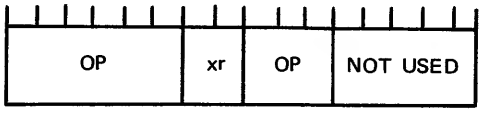
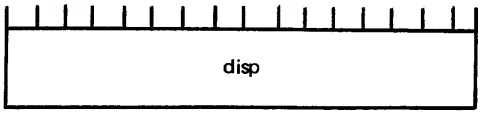
```
REVISION-C 11/20/72                                          73227   09143181
CUTIL    CONTROL PANEL AND TTY UTILITIES                     PAGE NUMBER    6
CONTROL PANEL ROUTINE
  306 FFE8 21D2 A          JMP ROUT
  307 FFE9         ;
  308 FFE9 A002 A SAVE:    ST 0,X'02     ;  SAVE AC0 - AC3 IN LOCATIONS X'02 - X'05.
  309 FFEA A403 A          ST 1,X'03
  310 FFEB A804 A          ST 2,X'04
  311 FFEC AC05 A          ST 3,X'05
  312 FFED 0080 A          PUSHF
  313 FFEE 4500 A          PULL 1
  314 FFEF A406 A          ST 1,X'06;       SAVE FLAGS IN LOCATION 6.
  315 FFF0 0200 A          RTS 0
  316 FFF1         ;


INTERRUPT SERVICE ROUTINE FOR HALT AND STACKFULL

  317 FFF1                 .PAGE          'INTERRUPT SERVICE ROUTINE FOR HALT AND STACKFULL'
  318 FFF1         ;
  319 FFF1 4CFF A INTR:    LI 0,-1;
  320 FFF2 1803 A          BOC C8,STFL;
  321 FFF3 29F5 A          JSR SAVE;      SAVE ACCUMULATORS IN LOCATIONS  2,3,4 AND 5.
  322 FFF4 4400 A          PULL 0
  323 FFF5 4000 A          PUSH 0
  324 FFF6 0600 A STFL:    ROUT 0;
  325 FFF7 0000 A ZERO:    HALT;
  326 FFF8 0400 A          RIN 0;         READ SWITCHES
  327 FFF9 1102 A          BOC C1,.+3;    IF SWITCHES ARE SET TO ALL ZEROS, 'EXECUTE' CAUSES
  328 FFFA 29D7 A          JSR RSTOR;     A RETURN TO THE CONTROL PANEL ROUTINE.
  329 FFFB 0100 A          RTI ;          IF SWITCHES ARE SET TO ANY NON-ZERO NUMBER,'EXECUTE
  330 FFFC         ;                      CAUSES A NORMAL RETURN FROM INTERRUPT.
  331 FFFC 4400 A          PULL 0
  332 FFFD 21BC A          JMP SET;       RETURN TO CONTROL PANEL ROUTINE.
  333 FFFE 21B6 A          JMP BEGIN
  334 FFFF 0006 A LAST6:   .WORD 6
  335  000                 .END

  ******      0 ERRORS IN ASSEMBLY   ******
```

(listing continued)

```
REVISION-C 11/20/72                                          73227   09143181
CUTIL    CONTROL PANEL AND TTY UTILITIES                     PAGE NUMBER   7
INTERRUPT SERVICE ROUTINE FOR HALT AND STACKFULL

$EXCL   $NU     $OUT    $XX     ABSTTY  ALFA    ALPHA   ASC     BEGIN   C1
FFAC A  FFAA A  FF85 A  FF5B A  FF00 A  FF7E A  FF8C A  FF92 A  FFB5 A  0001 A

C11     C12     C13     C14     C15     C2      C3      C4      C5      C7
0008 A  000C A  000D A  000E A  000F A  0002 A  0003 A  0004 A  0005 A  0007 A

C8      CRETRN  DELAY   DISP    DISPAC  ENDREC  EX      EXCLAM  EXITL1  FIVE
0008 A  FF8D A  FF38 A  FFE5 A  FFDE A  FF2B A  FFCF A  FF8F A  FF36 A  FFB3 A

H3FFF   H8000   INTR    JC14    JINTR   JSTRT   LA      LAST6   LD      LDAC
FF3C A  FF65 A  FFF1 A  000E A  FFB2 A  FFB1 A  FFC1 A  FFFF A  FFC8 A  FFE1 A

LINEFD  LOOP1   LOOP2   MSK1    MSKPAR  NINE    NUM     NUMBER  ONE     OUTT1
FF8E A  FF68 A  FF6B A  FF8A A  FF89 A  FF88 A  FF7F A  FF8B A  FFB4 A  FF9A A

PACK    PTBOOT  PTPNCH  PUT     RDPCK   RDWDCK  READR   RECV    REP     RETN
FF71 A  FF70 A  FF90 A  FF58 A  FF2F A  FF1C A  0003 A  FF3D A  FF49 A  FFA8 A

ROUT    RSRVE   RSTOR   SAVE    SEND    SET     START   STFL    TORS    TSTCKS
FF8B A  FFDA A  FFD2 A  FFE9 A  FF53 A  FFBA A  FF89 A  FFF6 A  FF24 A  FF1F A

TTY1    TTY2    V1      V2      WAIT    XMIT    ZERO
FF03 A  FF15 A  FF66 A  FF67 A  FFBC A  0004 A  FFF7 A


   8788     A617
```

Appendix B
FORMAT OF INSTRUCTIONS

A summary of the instruction types and their assembler language formats is given below for reference. A more-detailed breakdown of the instruction codes is shown in the next table; it is suitable for hand-coding small programs.

Instruction Type	Machine Format	Assembler Language Format		Remarks
Register to Register		Op	sr, dr	
Register to Memory		Op	r, disp	
Memory Reference (Class 1)		Op Op	r, disp(xr) r, @disp(xr)	Direct Indirect
Memory Reference (Class 2)		Op Op	disp(xr) @disp(xr)	Direct Indirect
I/O and Miscellaneous		Op	ctl	
Branch		Op	cc, disp	
Control Flags		Op	fc, ctl	
Memory Reference (Double Word)		Op	(xr)	
			disp	

Explanation of Symbols

Op – Instruction Mnemonic
OP – Operation Code
sr – Source Register Value
dr – Destination Register Value
xr – Index Register Value (2 or 3)
disp – Displacement Value
cc – Condition Code Value
r – Register Value
ctl – Control Bits Value

Table B–1. Basic Instruction Set with Bit Patterns

Mnemonic	Base
LD	8000
LD Indirect	9000
ST	A000
ST Indirect	B000
ADD	C000
SUB	D000
SKG	E000
SKNE	F000

Word Format

I = BASE ∨ r ∨ xr ∨ disp

r	REGISTER
0000	0
0400	1
0800	2
0C00	3

xr	ADDRESSING TECHNIQUE
0000	BASE PAGE
0100	PC RELATIVE
0200	INDEXED – AC2
0300	INDEXED – AC3

AND	6000
OR	6800
SKAZ	7000

r	REGISTER
0000	0
0400	1

ISZ	7800
DSZ	7C00
JMP	2000
JMP Indirect	2400
JSR	2800
JSR Indirect	2C00

BOC 1000

Word Format

I = BASE ∨ cc ∨ disp

Branch on	INTRPT=1 (when enabled)	AC0=0	AC0 ⩾ 0	AC0 ODD	AC0 Bit 1=1	AC0≠0	CPINT =1	START =1
CC	0000	0100	0200	0300	0400	0500	0600	0700

Branch on	STFL=1	INTEN=1	CYOV=1	AC0⩽0	USER	USER	USER	USER
CC	0800	0900	0A00	0B00	0C00	0D00	0E00	0F00

Word Format

I = BASE ∨ r ∨ disp

AISZ	4800
LI	4C00
CAI	5000
PUSH	4000
PULL	4400
XCHRS	5400
ROR/ROL	5800
SHR/SHL	5C00

LEFT DISP POSITIVE
RIGHT DISP NEGATIVE

r	REGISTER
0000	0
0100	1
0200	2
0300	3

Word Format

I = BASE ∨ sr ∨ dr

RADD	3000
RXCH	3080
RCPY	3081
RXOR	3082
RAND	3083

sr	dr	REGISTER
0000	0000	0
0400	0100	1
0800	0200	2
0C00	0300	3

Table B–1. Basic Instruction Set with Bit Patterns (Continued)

Mnemonic	Base	fc	FLAG	Word Format
SFLG	0800			I = BASE ∨ fc ∨ ctl
PFLG	0880	0000	8	
		0100	9	
		0200	10	
		0300	11	
		0400	12	
		0500	13	
		0600	14	
		0700	15	

						Word Format
						I = BASE ∨ ctl
HALT	0000	RTI	0100	RIN	0400	
PUSHF	0080	RTS	0200	ROUT	0600	
PULLF	0280	JSRI	0380	(Address range = FF80 to FFFF)		

The instruction is formed by the inclusive Or of each field. For example, the instruction RADD 2,3 is coded as X'3B00

For instructions that use the CTL field, only the first 7 bits (bits 0 through 6) are considered.

Examples of coding follow:

Example 1

RADD 2,3

BASE = 3000
sr = 0800
dr = 0300
INSTRUCTION = 3B00

Comments

Add AC2 to AC3.

Example 2

JMP–1 (3)

BASE = 2000
xr = 0300
disp = 00FF
INSTRUCTION = 23FF

Comments

Jump to the location specified by the index register AC3 modified by the displacement –1.

Example 3

SHR 0,1

BASE = 5C00
r = 0000
disp = 00FF
INSTRUCTION = 5CFF

Comments

Shift the contents of AC0 one place to the right.

Table B–2. Extended Instruction Set with Bit Patterns

Mnemonic	Base			Word Format
MPY	0480			I (Word 1) = BASE ∨ xr
DIV	0490			I (Word 2) = disp
DADD	04A0			
DSUB	04B0			
		xr	ADDRESSING TECHNIQUE	
		0000	Direct	
		0100	PC Relative	
		0200	Indexed – AC2	
		0300	Indexed – AC3	
				Word Format
LDB	04C0			I (Word 1) = BASE ∨ xr
STB	04D0			I (Word 2) = 2. disp ∨ Byte
				Byte = 0 for right, 1 for left.
ISCAN	0510			
				Word Format
SETST	0700			I = BASE ∨ Bit
CLRST	0710			
SETBIT	0720			
CLRBIT	0730			
SKSTF	0740			
SKBIT	0750			
CMPBIT	0760			
				Word Format
		BASADR	MAXADR	
JSRP	0300	(0100)	007F	I = BASE ∨ Incr
JMPP	0500	(0100)	000F	Incr = ADDR - BASADR
JINT	0520	0120	000F	BASADR ⩽ ADDR ⩽ BASADR + MAXADR

Example 1

LDB X'A0A

BASE = 04C0
xr = 0000
INSTRUCTION = 04C0 (Word 1)

disp = X'0A0A
Byte = 0 (Right)

INSTRUCTION = 1413 (Word 2)

Comments

Load AC0 with the right byte of location X'0A0A; direct addressing.

Example 2

JMPP 2

BASE = 0500
BASADR = (0100) Implied
INCR = 0002

INSTRUCTION = 0502

Comments

Jump through pointer located at location X'0102.

Appendix C
MEMORY ARRANGEMENT

The arrangement of memory in this system is shown in figure C–1. The entire base page consists of read/write memory; that is, locations $(0000)_{16}$ to $(00FF)_{16}$. The top sector of memory $(FE00)_{16}$ to $(FFFF)_{16}$ is implemented with read-only memory. The read-only memory on the board is enabled whenever bit 15 of the address is 1.

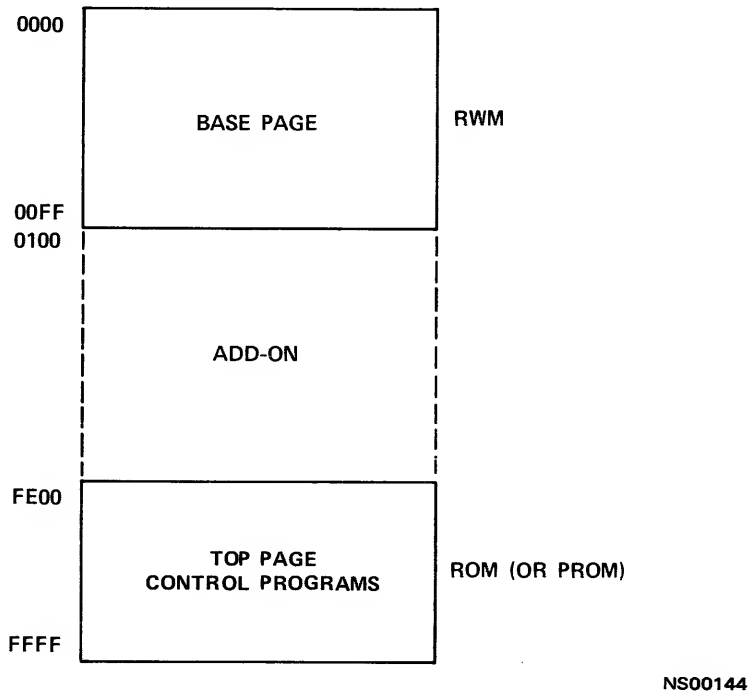


Figure C–1. Memory Layout

The memory arrangement shown above may be moved around to suit the convenience of the user. Jumper connections are provided on the IMP-16C card (W1, W2, and W3) to disable the on-board address decode logic. By removing these and using the external chip select inputs (CS0, CS1, and CS2), the user may generate his own decode signals to move the memory pages wherever desired.

Appendix D
IMP-16C NOMENCLATURE

Table D–1. Nomenclature Used in Circuit Schematics and Text

Signal Name	Description
ADEN	Address Enable Signal
ADX (0), (1), ..., (15)	Address Lines to Memory
ALU	Arithmetic/Logic Unit
BDO (0), (1), ..., (15)	Buffered Data-Out Lines
CI	Memory Cycle Initiate
CJMUX	Conditional Jump Multiplexer
CLK, CLK*	Master Clock Signals
CPINP	Control Panel Interrupt Acknowledge
CPINT	Control Panel Interrupt
CROM	Control Read-Only Memory
CSH0, CSH3	Carry/Shift Signal Lines (RALU)
CS0, CS1, CS2	Memory Chip-Select Lines
CYOV	Carry or Overflow Signal
C81, C23, C45, C67	Clock Signals, each lasting for two periods
DATA (0), (1), (2), and (3)	Data Bus Lines
DI (0), (1), (2), and (3)	Data Input Lines to the CROM
DISTR	Data Input Strobe
ENCTL	Enable Control Signal
START	Start or continue signal to restart operation
FLAG0, FLAG12	User status flags
F8, F11, ..., F15	User control flags
HCLK	Clock Hold Signal
HLT	Halt Flag
HOCSH	High-Order Carry/Shift Signal Line (CROM)
HOLD	Phase 4 Hold Signal
INIT*	Initialize Line (Complemented)
INTEN	Interrupt Enable Flag
INTRPT	Delayed Interrupt Signal
INTRA	Interrupt Request Signal
INTCTL	Interrupt Control Signal
JCSTR	Jump Condition Strobe Signal
JFA	Jump Flag Addresses

Table D–1. Nomenclature Used in Circuit Schematics and Text (Continued)

Signal Name	Description
LDAR	Load Address Register Flag
LOCSH	Low-Order Carry/Shift Signal Line (CROM)
MDO (0), (1), ..., (15)	Memory Data Out
MUX	Multiplexer
NCB (0), (1), (2), and (3)	Complemented Control Bus Lines
NFLEN	Flag Enable Signal Line (Complemented)
NJCOND	Jump Condition Input Line (Complemented)
NREQ0	Register Equal Zero Signal (Complemented)
POC	Power-On Condition
PROM	Electrically Programmed Read-Only Memory
RALU	Register/Arithmetic/Logic Unit
RDM	Read Memory Flag
RDM–Q1	Delayed Read Memory Flag
RDP	Read Peripheral Flag
RFREQ	Memory Refresh Request Signal
RFSH	Memory Refresh Initiate
SEL	Select Flag
SININ	Sign-In Signal Line
STF	Stack Full Signal Line
STFL	Stack Full Interrupt Signal
SVRST	Save/Restore Flag
SW	Switch Data (Input Port for Peripheral Data)
SYSCLR*	System Clear (Complemented)
V_{GG}, V_{SS}	Supply Voltages
WRM	Write Memory Flag
WRMP	Write Memory Pulse
WRP	Write Peripheral Flag
WRPA	Write Peripheral Strobe A
WRPB	Write Peripheral Strobe B
WRP3	Write Peripheral Strobe
PH1, PH3, PH5, and PH7	Clock Phase Times 1, 3, 5, and 7. (Each of these phase times corresponds to a clock pulse: T1, T3, T5, and T7.)

NOTE: An asterisk (*) after a signal name (except a CROM or a RALU signal) denotes a complemented signal. Complementation is denoted for CROM and RALU signals by the prefix N as part of the signal names.

Appendix E
LIST OF PIN CONNECTIONS AND SIGNALS ON IMP-16C CARD

Table E–1. IMP-16C Pin Numbers and Corresponding Signal Names

Pin Number	Signal Name	Pin Number	Signal Name
1	Ground	2	Ground
3	Ground	4	Ground
5	+5 volts	6	+5 volts
7	+5 volts	8	+5 volts
9	FLAG12 – Status Flag from RALU	10	ADX00 – Address Line, Bit 00
11	SVGG – Switched –12 volts[1]	12	SVGG – Switched –12 volts[1]
13	–10 volts for Read/Write Memory (MM1101A2)	14	–9 volts for Read/Write Memory (MM1101A2)
15	INTRA – Interrupt Request Signal	16	INIT* – Initialize (Complemented)
17	Not Used	18	ADX07 – Address Line, Bit 07
19	Not Used	20	ADX06 – Address Line, Bit 06
21	CS0 – Memory Chip-Select Line	22	FLAG0 – Status Flag from RALU
23	ADX02 – Address Line, Bit 02	24	ADX01 – Address Line, Bit 01
25	ADX03 – Address Line, Bit 03	26	Not Used
27	BDO08 – Buffered Data Out, Bit 08	28	ADX05 – Address Line, Bit 05
29	ADX04 – Address Line, Bit 04	30	C3B* – Timing Signal
31	–12 volts	32	–12 volts
33	Not Used	34	CS1 – Memory Chip-Select Line
35	CS2 – Memory Chip-Select Line	36	ODIS – Address Bus Disable
37	Not Used	38	Not Used
39	Not Used	40	MDO00 – Memory Data Out, Bit 00
41	MDO01 – Memory Data Out, Bit 01	42	Not Used
43	MDO02 – Memory Data Out, Bit 02	44	MDO03 – Memory Data Out, Bit 03
45	Not Used	46	Not Used
47	MDO04 – Memory Data Out, Bit 04	48	MDO05 – Memory Data Out, Bit 05
49	Not Used	50	MDO06 – Memory Data Out, Bit 06
51	Not Used	52	MDO07 – Memory Data Out, Bit 07
53	WRPB – Write Peripheral Strobe B	54	Not Used
55	Not Used	56	BDO05 – Buffered Data Out, Bit 05
57	DISTR* – Data Input Strobe (Complemented)	58	BDO01 – Buffered Data Out, Bit 01
59	BDO04 – Buffered Data Out, Bit 04	60	BDO00 – Buffered Data Out, Bit 00
61	BDO03 – Buffered Data Out, Bit 03	62	MDO14 – Memory Data Out, Bit 14
63	BDO07 – Buffered Data Out, Bit 07	64	BDO02 – Buffered Data Out, Bit 02
65	BDO06 – Buffered Data Out, Bit 06	66	WRPA – Write Peripheral Strobe A
67	BDO09 – Buffered Data Out, Bit 09	68	BDO11 – Buffered Data Out, Bit 11
69	BDO14 – Buffered Data Out, Bit 14	70	BDO15 – Buffered Data Out, Bit 15

1 - Should not be used on IMP-16/200 and 300 cards.

Table E–1. IMP-16C Pin Numbers and Corresponding Signal Names (Continued)

Pin Number	Signal Name	Pin Number	Signal Name
71	Ground	72	Ground
73	BDO10 – Buffered Data Out, Bit 10	74	BDO13 – Buffered Data Out, Bit 13
75	BDO12 – Buffered Data Out, Bit 12	76	MDO10 – Memory Data Out, Bit 10
77	MDO13 – Memory Data Out, Bit 13	78	MDO12 – Memory Data Out, Bit 12
79	SW00 – Switch Data, Bit 00	80	DSLCT – Input Data Select
81	SW02 – Switch Data, Bit 02	82	MDO08 – Memory Data Out, Bit 08
83	Not Used	84	SW01 – Switch Data, Bit 01
85	INTCTL – Interrupt Control	86	SW03 – Switch Data, Bit 03
87	MDO09 – Memory Data Out, Bit 09	88	MDO11 – Memory Data Out, Bit 11
89	WRM – Write Memory Flag	90	SW08 – Switch Data, Bit 08
91	SW10 – Switch Data, Bit 10	92	SW09 – Switch Data, Bit 09
93	SW11 – Switch Data, Bit 11	94	CLK81 – Timing Signal
95	SW12 – Switch Data, Bit 12	96	SW13 – Switch Data, Bit 13
97	SW14 – Switch Data, Bit 14	98	SW04 – Switch Data, Bit 04
99	C45 – Timing Signal	100	SW06 – Switch Data, Bit 06
101	ADX13 – Address Line, Bit 13	102	EXHOLD – External Hold
103	SW05 – Switch Data, Bit 05	104	JC14 – General-Purpose Jump Condition
105	SW07 – Switch Data, Bit 07	106	SW15 – Switch Data, Bit 15
107	JC15 – General-Purpose Jump Condition	108	MDO15 – Memory Data Out, Bit 15
109	ADX11 – Address Line, Bit 11	110	ADX10 – Address Line, Bit 10
111	ADX08 – Address Line, Bit 08	112	ADX09 – Address Line, Bit 09
113	RDM – Read Memory Flag	114	ADX12 – Address Line, Bit 12
115	ADX14 – Address Line, Bit 14	116	ADX15 – Address Line, Bit 15
117	CPINP – Control Panel Interrupt	118	CPINT – Control Panel Interrupt
119	JC13 – General-Purpose Jump Condition	120	WRMP – Write Memory Pulse
121	RDP – Read Peripheral Flag	122	WRP – Write Peripheral Flag
123	START – Start Signal	124	RFREQ – Memory Refresh Request
125	HLT* (FLAG)	126	SYSCLR* – System Clear (Complemented)
127	CLK* – Master Clock (Complemented)	128	CLK – Master Clock
129	JC12 – General-Purpose Jump Condition	130	F8 – General-Purpose User Flag
131	INT EN – Interrupt Enable Flag	132	F15 – General-Purpose User Flag
133	F11 – General-Purpose User Flag	134	F14 – General-Purpose User Flag
135	F13 – General-Purpose User Flag	136	F12 – General-Purpose User Flag
137	+5 volts	138	+5 volts
139	+5 volts	140	+5 volts
141	Ground	142	Ground
143	Ground	144	Ground

NOTE: 1. Odd-numbered pins are on component side and even-numbered pins are on solder side of IMP-16C card.
2. Pins 83 and 55 are used for CYCLE INITIATE and EXTERNAL REFRESH functions, respectively, in the IMP-16P system (IMP-16P/200, /300).

National Semiconductor Corporation
2900 Semiconductor Drive
Santa Clara, California 95051
(408) 732-5000
TWX: 910-339-9240

National Semiconductor Electronics SDNBHD
Batu Berendam
Free Trade Zone
Malacca, Malaysia
Telephone: 5171
Telex: NSELECT 519 MALACCA (c/o Kuala Lumpur)

National Semiconductor GmbH
D 808 Fuerstenfeldbruck
Industriestrasse 10
West Germany
Telephone: (08141) 1371
Telex: 27649

National Semiconductor (UK) Ltd.
Larkfield Industrial Estates
Greenock, Scotland
Telephone: (0475) 33251
Telex: 778 632

National Semiconductor (Pte.) Ltd.
No. 1100 Lower Delta Rd.
Singapore 3
Telephone: 630011
Telex: 21402

REGIONAL AND DISTRICT SALES OFFICES

ALABAMA
DIXIE DISTRICT OFFICE
3322 Memorial Pkway, S.W. #67
Huntsville, Alabama 35802
(205) 881-0622
TWX: 810-726-2207

ARIZONA
*ROCKY MOUNTAIN REGIONAL OFFICE
3313 North 68th Street, No. 114
Scottsdale, Arizona 85251
(602) 945-8473

CALIFORNIA
*NORTH-WEST REGIONAL OFFICE
2680 Bayshore Frontage Road, Suite 112
Mountain View, California 94043
(415) 961-4740
TWX: 910-379-6432

NATIONAL SEMICONDUCTOR
*DISTRICT SALES OFFICE
Valley Freeway Center Building
15300 Ventura Boulevard, Suite 305
Sherman Oaks, California 91403
(213) 783-8272
TWX: 910-495-1773

NATIONAL SEMICONDUCTOR
SOUTH-WEST REGIONAL OFFICE
17452 Irvine Boulevard, Suite M
Tustin, California 92680
(714) 832-8113
TWX: 910-595-1523

CONNECTICUT
AREA OFFICE
Commerce Park
Danbury, Connecticut 06810
(203) 744-2350

*DISTRICT SALES OFFICE
25 Sylvan Road South
Westport, Connecticut 06880
(203) 226-6833

FLORIDA
*AREA SALES OFFICE
2721 South Bayshore Drive, Suite 121
Miami, Florida 33133
(305) 446-8309
TWX: 810-848-9725

CARIBBEAN REGIONAL SALES OFFICE
P.O. Box 6335
Clearwater, Florida 33518
(813) 441-3504

ILLINOIS
NATIONAL SEMICONDUCTOR
WEST-CENTRAL REGIONAL OFFICE
800 E. Northwest Highway, Suite 203
Mt. Prospect, Illinois 60056
(312) 394-8040
TWX: 910-689-3346

INDIANA
NATIONAL SEMICONDUCTOR
NORTH-CENTRAL REGIONAL OFFICE
P.O. Box 40073
Indianapolis, Indiana 46240
(317) 255-5822

KANSAS
DISTRICT SALES OFFICE
13201 West 82nd Street
Lenexa, Kansas 66215
(816) 358-8102

MARYLAND
CAPITAL REGIONAL SALES OFFICE
300 Hospital Drive, No. 232
Glen Burnie, Maryland 21061
(301) 760-5220
TWX: 710-861-0519

MASSACHUSETTS
*NORTH-EAST REGIONAL OFFICE
No. 3 New England, Exec. Office Park
Burlington, Massachusetts 01803
(617) 273-1350
TWX: 710-332-0166

MICHIGAN
*DISTRICT SALES OFFICE
23629 Liberty Street
Farmington, Michigan 48024
(313) 477-0400

MINNESOTA
DISTRICT SALES OFFICE
8053 Bloomington Freeway, Suite 101
Minneapolis, Minnesota 55420
(612) 888-3060
Telex: 290766

NEW JERSEY/NEW YORK CITY
MID-ATLANTIC REGIONAL OFFICE
301 Sylvan Avenue
Englewood Cliffs, New Jersey 07632
(201) 871-4410
TWX: 710-991-9734

NEW YORK (UPSTATE)
CAN-AM REGIONAL SALES OFFICE
104 Pickard Drive
Syracuse, New York 13211
(315) 455-5858

OHIO/PENNSYLVANIA/
W. VIRGINIA/KENTUCKY
EAST-CENTRAL REGIONAL OFFICE
Financial South Building
5335 Far Hills, Suite 214
Dayton, Ohio 45429
(513) 434-0097

TEXAS
*SOUTH-CENTRAL REGIONAL OFFICE
5925 Forest Lane, Suite 205
Dallas, Texas 75230
(214) 233-6801
TWX: 910-860-5091

WASHINGTON
DISTRICT OFFICE
300 120th Avenue N.E.
Building 2, Suite 205
Bellevue, Washington 98005
(206) 454-4600

INTERNATIONAL SALES OFFICES

AUSTRALIA
*NATIONAL SEMICONDUCTOR
ELECTRONICS PTY, LTD.
Cnr. Stud Road & Mountain Highway
Bayswater, Victoria 3153
Australia
Telephone: 729-0733
Telex: 32096

CANADA
*NATIONAL SEMICONDUCTOR CORP.
1111 Finch Avenue West
Downsview, Ontario, Canada
(416) 635-9880
TWX: 610-492-2510

DENMARK
NATIONAL SEMICONDUCTOR
SCANDINAVIA
Vordingborggade 22
2100 Copenhagen
Denmark
Telephone: (01) 92-OBRO-5610
Telex: DK 6827 MAGNA

ENGLAND
NATIONAL SEMICONDUCTOR (UK) LTD.
The Precinct
Broxbourne, Hertfordshire
England
Telephone: 69571
Telex: 267-204

FRANCE
NATIONAL SEMICONDUCTOR
FRANCE S.A.R.L.
28, Rue de la Redoute
92260-Fontenay-Aux-Roses
Telephone: 660-81-40
TWX: NSF 25956F

HONG KONG
*NATIONAL SEMICONDUCTOR
HONG KONG LTD.
9 Lai Yip Street
Kwun Tung, Kowloon
Hong Kong
Telephone: 3-458888
Telex: HX3866

JAPAN
*NATIONAL SEMICONDUCTOR JAPAN
Nakazawa Building
1-19 Yotsuya, Shinjuku-Ku
Tokyo, Japan 160
Telephone: 03-359-4571
Telex: J 28592

SWEDEN
NATIONAL SEMICONDUCTOR SWEDEN
Sikvagen 17
13500 Tyreso
Stockholm
Sweden
Telephone: (08) 712-04-80

WEST GERMANY
*NATIONAL SEMICONDUCTOR GMBH
8000 Munchen 81
Cosimstrasse 4
Telephone: (0811) 915-027

***Microprocessor System Specialist Available**

CP10 M34